

Norman Balabanian
Bradley Carlson

PRINCIPIOS DE

Diseño Lógico Digital

CECSA

PRINCIPIOS DE

Diseño Lógico Digital

Esta obra ofrece una clara introducción a los principios del diseño lógico digital. A diferencia de otras obras similares, la mayor parte de los temas se presentan desde una perspectiva exploratoria a semejanza de un proyecto de investigación, cuyo objetivo consiste en descubrir y asimilar conocimientos sobre el tema expuesto. Así, al abordar un tema nuevo, siempre se intenta que los estudiantes comprendan su importancia y se involucren en el descubrimiento de los conceptos.

Características

- Se incluyen abundantes ilustraciones, ejemplos y ejercicios para cada tema con los cuales el estudiante puede reforzar y aplicar los conceptos vistos en la obra.
- Al final de cada capítulo, el lector encontrará una serie de problemas que van desde una aplicación muy sencilla de los procedimientos formulados en el texto hasta la solución de otros más complejos y de mayor dificultad.
- Se ha elegido el lenguaje ABEL para presentar los lenguajes de descripción en hardware (HDL) como una herramienta de diseño. Con esto se reduce el esfuerzo de los estudiantes para aprender el lenguaje y les permite concentrarse en los conceptos subyacentes al diseño con un HDL.

La obra está dirigida a estudiantes de los primeros años de ingeniería eléctrica, electrónica y en ciencias computacionales.



COMPAÑÍA EDITORIAL CONTINENTAL

ISBN 970-24-0256-5



9 789702 402565

Principios de diseño lógico digital

621.395
B18 P
1ed
E.1

Principios de Diseño Lógico Digital

Norman Balabanian
Universidad de Florida

Bradley Carlson
Symbol Technologies, Inc.

k-t-dra
Grupo k-t-dra

Diagonal 85A Ito 26-05 Polo Club • Fax (57) 1 2187629
Teléfonos 2570895 • 6358137 • A.A. 93825 • Bogotá D.T.
Colombia • e-mail: info@k-t-dra

www.k-t-dra.com

UNIVERSIDAD DE LA SALLE
BIBLIOTECA P.T.

PRIMERA EDICIÓN
MÉXICO, 2002

47964

COMPAÑÍA EDITORIAL CONTINENTAL

Para establecer comunicación
con nosotros puede hacerlo por:



correo:
Renacimiento 180, Col. San Juan
Tlihuaca, Azcapotzalco,
02400, México, D.F.



fax pedidos:
(015) 561 4063 • 561 5231



e-mail:
Info@patriacultural.com.mx



home page:
<http://www.patriacultural.com.mx>

Título original de la obra:

Digital Logic Design Principles / Norman Balabanian, Bradley Carlson

ISBN: 0-471-29351-2

Copyright © 2001, John Wiley & Sons, Inc. All Rights Reserved

Traducción autorizada de la edición en inglés publicada por
John Wiley & Sons, Inc.

Dirección editorial: Javier Enrique Callejas

Coordinadora editorial: Elisa Pecina Rosas

Diseño de interiores: Guillermo Rodríguez Luna

Diseño de portada: Perla López Romo

Traducción:

Ing. Gabriel Nagore Cázares

Revisión técnica:

Dr. Cuauhtémoc Carbajal Fernández

Profesor investigador del Depto. de Eléctrica y Electrónica

ITESM-Campus Estado de México

Principios de diseño lógico digital

Derechos reservados respecto a la edición en español:

© 2002, GRUPO PATRIA CULTURAL, S.A. DE C.V.

bajo el sello de Compañía Editorial Continental

Renacimiento 180, Colonia San Juan Tlihuaca,

Delegación Azcapotzalco, Código Postal 02400, México, D.F.

Miembro de la Cámara Nacional de la Industria Editorial

Registro núm. 43

ISBN 970-24-0256-5

Queda prohibida la reproducción o transmisión total o parcial del conte-
nido de la presente obra en cualesquiera formas, sean electrónicas o
mecánicas, sin el consentimiento previo y por escrito del editor.

Impreso en México

Printed in Mexico

Primera edición: 2002

Esta obra se terminó de imprimir en marzo del 2002
en los talleres de Litográfica INGRAMEX, S.A. de C.V.
Centeno No. 162 Local 1, Col. Granjas Esmeralda
C.P. 09810, México, D.F.

Prefacio

EL LIBRO

Éste es un libro de nivel introductorio sobre los *principios* de diseño lógico digital. Se dirige a estudiantes de primero y segundo año de ingeniería eléctrica, ingeniería electrónica, ingeniería en computación o ciencias computacionales. No se requieren conocimientos previos de circuitos eléctricos o de electrónica. También lo encontrarán útil los lectores que necesiten abordar por primera vez o revisar los principios del diseño digital.

Aspectos pedagógicos

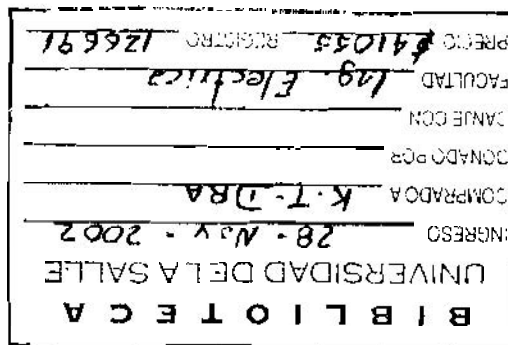
El proceso deductivo —la aplicación de principios generales a casos específicos— suele ilustrarse adecuadamente en los libros de **texto**. A menudo, el **autor** anuncia un concepto general o un aspecto de un tema o **resultado**, seguido de ejemplos de aplicación del concepto. Cuando los **estudiantes** inician un tema, **no tienen muy** claras las ideas que **motivan una definición** o un procedimiento general. Tampoco comprenden la utilidad o interés que **reviste** un tema, o su trascendencia.

En este libro, **adoptamos** un enfoque inductivo en la **presentación** del material, que incluye la formulación de un resultado generalmente **válido** a partir del estudio de casos específicos, como el **procedimiento** que se seguiría en un **proyecto** de **investigación**. Un investigador **llega** a un resultado por lo general **válido** después de **varios experimentos** o **cálculos** específicos. Algunas veces el estudio de uno o más casos específicos conduce a una conjetura generalmente válida. **Posteriormente**, la conjetura se analiza y justifica utilizando resultados establecidos con anterioridad.

De **modo** similar, presentamos la mayor parte de los **temas** desde una perspectiva exploratoria, en **vez** de ofrecerlos al lector sin ninguna justificación. La exposición del texto se asemeja a un proyecto de investigación, cuyo **objetivo** consiste en descubrir y asimilar **conocimientos** sobre la terna que se estudia. **Al** abordar un **tema**, se realiza un esfuerzo considerable para contribuir a que los **estudiantes entiendan** por qué le debemos dedicar tiempo. Una vez **que** se agota un **tema** (esto es, cuando necesitamos **pasar** al siguiente), se analizan las alternativas: "Podemos **hacer esto o aquello**", y el **comentario** podría **proseguir**: "intentemos **primero lo siguiente**, por las siguientes razones". Por **qué** seguir un hilo particular y **cómo** podría presentarse un **procedimiento**, son **cuestiones** tan importantes como aclarar al estudiante los detalles del procedimiento o de la aplicación de cierto algoritmo.

Cuando un **tema**, como el de los circuitos **digitales**, alcanza un nivel avanzado un libro de texto tiende a adquirir **características** enciclopédicas: se **aborda** todo lo concebible. **Este** enfoque oculta al estudiante el juego del descubrimiento. **Se le brinda** la historia completa y se le indica qué aprender de ella, practicando con los ejercicios y problemas **planteados** en el libro. En **este texto**, tratamos de evitar el error de catalogar todo lo que **sabemos** sobre el tema. En los problemas planteados en el texto, dejamos a los estudiantes el placer de generar (de manera guiada) **resultados** que **no** son esenciales para continuar con el **tema**, por lo que no es necesario que **formen parte** de la exposición.

Sabemos que los estudiantes aprenden mejor si están **comprometidos**. No hay mucho **que** los autores puedan hacer **para mantenerlos** así, aunque insistimos en que participen **en** la deducción de una ecuación completando pasos omitidos, pidiéndoles **que observen** los rasgos **relevantes** de un diagrama o tabla que **ellos describen** de manera cuidadosa, o solicitándoles que **analicen** un plan propuesto antes de llevarlo a cabo con detalle. **Con** frecuencia recurrimos a estos **procedimientos**.



Nivel de presentación

El material de este libro es **introdutorio**, para **primero** o segundo año de universidad. Sin **em-**bargo, el nivel de un libro no debe dictar el grado de **rigor** en la presentación. Todo tema del libro se trata rigurosamente.

Selección de temas

La **selección** de los temas fue la usual. La **selección** y el orden de los temas facilita el uso del libro en instituciones con diferentes calendarios y **una** diversidad de enfoques. El libro puede utilizarse en cursos que abarcan un **año** académico, ya sea de dos semestres o tres **cuatrimes-**tres, especialmente si se **presta atención** a la sección relativa al laboratorio (vea la descripción del manual del laboratorio). Mediante la **selección** adecuada de capítulos y temas de **cada** capítulo, es posible adaptar un curso de un semestre. Los temas "**suplementarios**" se **presentan** en secciones que los profesores pueden omitir sin que incurran en una **falta**. Las secciones o problemas finales que se basan en este material se pueden omitir **también** si se desea. La **inclusión** de material de este tipo **permite** a los estudiantes con **más** tiempo o **interés** beneficiarse sin **afec-**tar a los **demás**.

La **elección** de **ABEL** para aprender un lenguaje de descripción de hardware (HDL) como **herramienta** de **diseño**, reduce el esfuerzo de los estudiantes para aprenderse el lenguaje, lo que les permite concentrarse en los conceptos de diseño con un HDL. Todos los conceptos de la especificación, **simulación** y **síntesis** HDL pueden enseñarse utilizando ABEL; así, el estudiante no tiene necesidad de aprenderse la sintaxis y la **semántica** de un lenguaje complejo como VHDL o Verilog.

Esquema de numeración de ecuaciones y figuras

En ocasiones es posible que algunos **esquemas** de numeración de secciones, ecuaciones y figuras, así como las remisiones a **ellas**, distraigan al **estudiante** que dedica tiempo de manera **improduc-**tiva a la búsqueda y lectura de los números. En este libro **utilizamos** un sistema de **numeración** **secuencial**, que se inicia en cada **capítulo**, tanto para las ecuaciones como para las figuras. (Cuando se haga referencia a una ecuación de un capítulo anterior, también se indica el número de capítulo.) De manera similar, las secciones **principales** de un capítulo se enumeran de manera consecutiva, sin indicar el capítulo, **pero** las secciones secundarias y terciarias no se **enumé-**ran; así se evita la lectura **improductiva** de números de sección tales como 4.3-5, que **identifican** la subsección 5 de la sección 3 del capítulo 4. Es rara la referencia a una de estas subsecciones **particulares**. si es que **se llega** a hacer en algún libro; por consiguiente, no se **otorga** valor alguno a un esquema de **numeración** de este tipo. No se enumeran todas las **ecuaciones**, sino sólo las importantes o a las **que se hace referencia** más tarde. Cuando nos referimos a una **ecuación** o a una figura, la **indicamos** con el término **ecuación** o **figura**.

Ilustraciones, ejemplos, ejercicios y problemas

Al explicar un **tema**, se emplean ilustraciones para **aclararlo**. En **realidad**, una ilustración podría preceder a la explicación del tema como parte del proceso de **inducción**. Las **ilustraciones** se incorporan de ese modo en la presentación del material. **También hay ejemplos** numerados, separados del texto y fácilmente distinguibles, los **cuales** se abordan utilizando los conceptos que acaban de **plantearse**, junto con otras ideas recientemente **asimiladas**.

A lo largo de la exposición y con un formato **que los distingue** del **texto**, se encuentran los **ejercicios** numerados que los estudiantes deben resolver en el **tiempo** en que estudian las **seccio-**

nes **importantes**. El objetivo de estos ejercicios consiste en reforzar los conceptos que se estudian, invitando a los estudiantes a que **efectúen** algunos cálculos simples y **apliquen** después los resultados explicados. Éstos forman parte de la idea del *proyecto de investigación*. Los requerimientos de excitación para un tipo de flip-flop podrían formularse dentro del **texto**, por ejemplo: los requerimientos de excitación para otros tipos de flip-flops se dejan como ejercicio a los **estudiantes**. Cuando resulte de utilidad, se ofrecen las respuestas a fin de que los estudiantes confirmen los **resultados** de sus esfuerzos. (La mayoría de las **veces**, en especial si las **respuestas** son breves y por ello fáciles para que los estudiantes les den un vistazo dentro del texto, éstas se presentan al pie de las páginas.) Los ejercicios **no sólo requieren** la **repetición** de los pasos de un ejemplo analizado, cambiando valores o configuraciones de circuito. Por consiguiente, no hay necesidad de ofrecer **ejemplos** analizados antes de solicitar a los estudiantes que resuelvan un ejercicio.

Al final de cada capítulo se incluye una serie de *problemas*. Los problemas de cada serie van desde la **simple** aplicación de **procedimientos** formulada en el libro, hasta la solución de problemas más **complejos**, o de gran utilidad. Algunas veces un problema requiere que los **estudiantes** apliquen una técnica específica. En otros casos se les pide que resuelvan un problema utilizando dos o más enfoques y que comparen el grado de dificultad. En **ambos casos**, practican técnicas específicas y refuerzan el dominio de ellas. A **veces** el problema es **abierto** de manera que los **estudiantes** tomen decisiones en torno a los métodos que **aplicarán**.

Complementos del texto

Hay dos paquetes de **complementos**. Uno **se facilita** a los profesores **que** adoptan el libro en sus cursos, el **cual** no se encuentra disponible para los **estudiantes**. Incluye un manual que contiene las soluciones completas de los **problemas** en el libro. **También** incluye un **conjunto** de transparencias de figuras de la obra. Éstas se amplían de manera que los profesores tengan la posibilidad de utilizarlas en el salón de clases.

El otro paquete **consta** de un manual de laboratorio. Aunque en ocasiones el libro se hace referencia a familias específicas de circuitos digitales (por ejemplo, 74LS02), el **interés** principal se centra en los principios de diseño. El **manual** de laboratorio persigue involucrar a los **estudiantes** en la *práctica* del **diseño** digital, utilizando lo **último** en la tecnología de que se dispone en la actualidad. En algunas **partes** del libro, indicamos la forma de incorporar proyectos de **diseño** **específicos** del manual. Aunque algunos estudiantes quizá aprendan diseño digital con otros textos, también pueden utilizar este manual de laboratorio para adquirir experiencia en la *práctica* del diseño digital. Si se desea **más** información relativa al manual del laboratorio, **visítese** el sitio web del texto (<http://www.wiley.com/college/elec/balabanian293512>).

SOFTWARE

Recomendamos desde el principio el uso de entradas esquemáticas, así como de simulación temporal y funcional en el laboratorio (incluso con **experimentos** o **prácticas** de laboratorio simples). Se puede utilizar el software Xilinx WebPack, el cual se obtiene **gratuitamente** en el **sitio web** Xilinx (<http://www.xilinx.com>). Este software apoya la versión más reciente de ABEL, de modo **que** al llegar al capítulo 8 los **estudiantes** estarán **familiarizados** con la **interfaz** de usuario.

RECONOCIMIENTOS

Quisiéramos dar nuestro agradecimiento a varias personas que han contribuido de diversas maneras a la **realización** de este libro. Norman Balabanian desea agradecer al doctor Vijay Pitchumani (ahora con Intel) y al doctor Dikran Meliksetian (ahora con IBM), de la Universidad de

Siracuse. En diferentes etapas fueron coautores de este libro e hicieron importantes contribuciones en la creación del texto.

Algunas de las personas que ofrecieron comentarios y observaciones invaluableles cuando se revisó el manuscrito en diferentes etapas de su creación fueron:

Yu Hen Hu, University of Wisconsin-Madison
David R. Kaeli, Northeastern University
Juanita DeLoach, University of Wisconsin-Milwaukee
Mehmet Celenk, Ohio University
James G. Harris, California Polytechnic State University, San Luis Obispo
Sotirios G. Ziavras, New Jersey Institute of Technology
James H. Aylor, University of Virginia
Ward D. Getty, University of Michigan, Ann Arbor
Alexandros Eleftheriadis, Columbia University in the City of New York
Ike Evans, The University of Iowa y Evolutionary Heuristics
Shahram Latifi, University of Nevada, Las Vegas
Gregory B. Lush, University of Texas en El Paso

Por último, deseamos agradecer a Ko-Chi Kuo, quien elaboró las soluciones de los problemas de fin de capítulo y redactó el manual de soluciones.

Contenido

Capítulo 1. REPRESENTACIÓN DE NÚMEROS, CÓDIGOS Y CONVERSIÓN DE CÓDIGOS 1

1. Sistemas digitales y analógicos 1
2. Hardware, software y firmware 3
3. Sistemas numéricos 4
 - El sistema binario y otros sistemas numéricos* 5
 - Conversiones de base* 6
 - Conversión al sistema decimal 6
 - Conversión a partir del sistema decimal 7
 - Del octal o hexadecimal al binario 8
 - Aritmética binaria* 9
 - Suma 9
 - Resta 10
 - Multiplicación 10
 - División 10
 - Complementos: a dos y a uno* 11
 - Suma de números binarios* 13
4. Códigos y conversión de código 15
 - Decimal codificado en el sistema binario* 16
 - Códigos ponderados* 16
 - Código Gray* 18
 - Código de siete segmentos* 18
 - Códigos alfanuméricos* 19
5. Detección y corrección de errores 20
 - Códigos de detección de errores* 21
 - Códigos de corrección de errores* 22
 - Códigos de Hamming* 23
- Resumen y repaso del capítulo 25
- Problemas 26

Capítulo 2. ÁLGEBRA DE CONMUTADORES Y COMPUERTAS LÓGICAS 32

1. **Álgebra booleana** 32
 - Principio de dualidad* 33
 - Teoremas fundamentales* 34
 - Álgebra de conmutación* 37
2. **Operaciones de conmutación** 38
 - La operación AND* 38
 - La operación OR* 39
 - La operación NOT* 39
 - Comentario* 39
3. **Expresiones de conmutación** 40
 - Minitérminos, maxitérminos y formas canónicas* 41
 - Generalización de la ley de De Morgan* 43
4. **Funciones de conmutación** 45
 - Operaciones de conmutación sobre funciones de conmutación* 46
 - Número de términos en formas canónicas* 47
 - Teorema de expansión de Shannon* 47
 - Forma de suma de productos* 48
 - Forma de producto de sumas* 49
5. **Otras operaciones de conmutación** 49
 - OR exclusiva* 50
 - Operaciones NAND, NOR y XNOR* 50
6. **Conjuntos de operaciones universales** 51
7. **Compuertas lógicas** 52
 - Formas alternativas de las compuertas NAND y NOR* 53
 - Compuertas OR exclusivas* 54
 - Comentario* 55
8. **Lógica positiva, negativa y combinada** 55
9. **Algunas cuestiones prácticas relativas a compuertas** 57
 - Familias lógicas* 58
 - Características de entrada/salida de compuertas lógicas* 59
 - Factor de carga de salida y factor de carga de entrada* 61
 - Búfers o reforzadores* 63
 - Consumo de potencia* 63
 - Margen de ruido* 64
 - Velocidad y retardo de propagación* 64
10. **Circuitos integrados** 66
 - Algunas características de los CI* 66
 - Economía de diseño* 68
 - CI de aplicación específica* 69
11. **Lógica alambrada** 69
 - Compuertas lógicas de tres estados (alta impedancia)* 69
 - Compuertas lógicas de colector abierto y de drenaje abierto* 70
- Resumen y repaso del capítulo** 71
- Problemas** 72

Capítulo 3. REPRESENTACIÓN E IMPLEMENTACIÓN DE FUNCIONES LÓGICAS 76

1. Listas de minitérminos y maxitérminos 76
 - Listas de minitérminos y forma de suma de productos* 77
 - Listas de maxitérminos y formas de producto de sumas* 78
2. Mapas lógicos 79
 - Adyacencia lógica y adyacencia geométrica* 79
 - Cubos de orden k* 84
3. Realizaciones mínimas de funciones de conmutación 87
 - Expresiones irreducibles y mínimas* 87
 - Implicantes primos* 88
 - Expresiones mínimas de suma de productos* 89
 - Expresiones mínimas de producto de sumas* 91
 - Implementaciones de dos niveles* 92
 - Implementación AND-OR 92
 - Implementación NAND 93
 - Implementación OR-AND 94
4. Implementación de expresiones lógicas 94
 - Análisis* 97
 - Características de circuitos de compuerta* 97
5. Diagramas de temporización 98
6. Funciones incompletamente especificadas 100
 - Valores irrelevantes* 100
7. Comparadores 102
 - Comparadores de 2 bits* 102
 - Generalización* 104
 - Comparadores de 4 bits* 104
 - Comparadores de números pares de bits* 105
 - Comparadores de números impares de bits* 105
8. Determinación del implicante primo: método tabular 105
 - Representaciones de cubos k adyacentes* 106
 - Clasificación por índice* 107
 - Funciones incompletamente especificadas* 109
 - Selección de una expresión mínima* 109
 - Funciones completamente especificada%* 109
 - Manejo de valores irrelevantes* 112
9. Circuitos de salida múltiple 112
- Resumen y repaso del capítulo 113
- Problemas 114

Capítulo 4. DISEÑO LÓGICO COMBINATORIO 125

1. Sumadores binarios 125
 - Sumador completo* 126
 - Sumador de acarreo propagado* 128
 - Sumador de acarreo anticipado* 128
 - Restador binario* 132
 - Sumador y restador de complemento a dos 132
 - Sumador y restador de complemento a uno 133
2. Multiplexores 134
 - Multiplexores como circuitos lógicos de propósito general* 136
3. Decodificadores y codificadores 139
 - Desmultiplexores* 139
 - Decodificador de n a 2^n líneas* 139
 - Decodificador de árbol* 141
 - Decodificadores como circuitos lógicos de propósito general: conversión de código* 142
4. Memoria sólo de lectura (ROM) 143
5. Otros dispositivos lógicos programables LSI 146
 - Arreglo lógico programado (PLA)* 146
 - Lógica de arreglo programado (PAL)* 148
- Resumen y repaso del capítulo 150
- Problemas 151

Capítulo 5. COMPONENTES DE CIRCUITOS SECUENCIALES 159

1. Definiciones y conceptos básicos 159
2. Cerrojos y flip-flops 162
 - Cerrojos SR* 163
 - Problemas de temporización y cerrojos SR con reloj* 166
 - Cerrojo JK* 168
 - Cerrojo maestro-esclavo* 168
 - Un diseño posible 169
 - Un diseño maestro-esclavo alternativo 170
 - Parámetros de activación por pulso 171
 - Flip-flops de retardo (D)* 171
 - Flip-flop *D* activado por flanco 172
 - Flip-flop T* 174
 - Requerimientos de excitación del flip-flop* 175
3. Registros 176
 - Registro de corrimiento de carga en serie* 176
 - Registro de corrimiento de carga en paralelo* 177
 - Conversión paralelo serie* 178
 - Registros universales* 180
- Resumen y repaso del capítulo 181
- Problemas 182

Capítulo 6. MÁQUINAS SECUENCIALES SÍNCRONAS 187

1. Conceptos básicos 187
 - Diagrama de estados* 189
 - Tabla de estados* 191
 - Construcción de una tabla de estados a partir de un diagrama de estados 192
2. Asignaciones de estado 194
 - Análisis* 196
 - Reglas prácticas para asignar estados* 198
3. Procedimiento de diseño general 201
 - Máquina de Mealy* 201
 - Máquina de Moore* 206
4. Equivalencia de estados y minimización de máquina 207
 - Distinguibilidad y equivalencia* 208
 - Minimización de máquina* 209
5. Máquinas con rangos de memoria finita 211
 - Máquinas con memoria de entrada finita* 211
 - Máquinas con memoria de salida finita* 213
 - Máquinas de memoria finita* 214
6. Contadores síncronos 215
 - Contadores de modo simple* 215
 - Contadores de distancia unitaria* 216
 - Contadores de anillo* 217
 - Estados indeterminados* 218
 - Contadores multimodo* 219
 - Contador ascendente-descendente de módulo 6* 221
7. Máquinas de estado algorítmicas 221
 - Principios básicos* 221
8. Entradas asíncronas 226
 - Comunicación asíncrona (protocolo de "apretón de manos")* 226
- Resumen y repaso del capítulo 228
- Problemas 229

Capítulo 7. MÁQUINAS SECUENCIALES ASÍNCRONAS 241

1. El modelo del modo fundamental 242
2. La tabla de flujo 243
 - Tablas de flujo primitivas* 243
 - Asignación de salidas para estados inestables* 246
3. Reducción de máquinas incompletamente especificadas 247
 - La tabla de fusión* 248
 - Compatibilidad* 248
 - Construcción de la tabla de fusión* 249
 - Determinación de coberturas cerradas mínimas* 251
 - Tablas de transición* 253
4. Carreras y ciclos 256
 - Carreras críticas y no críticas* 257
 - Ciclos y oscilaciones* 259
5. Riesgos 261
 - Riesgos estáticos* 261
 - Riesgos dinámicos* 265
 - Riesgos esenciales* 265
- Resumen y repaso del capítulo 267
- Problemas 268

Capítulo 8. DISEÑO CON LENGUAJES DE DESCRIPCIÓN DE HARDWARE 275

1. El lenguaje de descripción del hardware ABEL 276
 - Especificación del sumador en ABEL* 277
 - Descripción de comportamiento contra descripción operacional* 278
 - Especificación del sumador en ABEL* 280
 - Especificación de circuito secuencial en ABEL* 282
 - Condiciones irrelevantes en ABEL* 285
 - Especificaciones jerárquicas en ABEL* 286
2. Dispositivos lógicos programables (PLD) 290
 - Dispositivos lógicos programables complejos* 294
 - Arreglos de compuertas programable en campo* 299
3. El flujo de diseño para las especificaciones HDL 301
 - Síntesis y asignación de tecnología de especificaciones ABEL* 302
 - Simulación de especificaciones ABEL* 304
- Resumen y repaso del capítulo 306
- Problemas 306

Capítulo 9. ORGANIZACIÓN DE LA COMPUTADORA 308

1. Unidades de control y de trayectoria de datos de un procesador 308
 - Unidad de trayectoria de datos* 309
 - Unidad de control* 310
 - Ejemplo de multiplicador en serie 310
2. Computadora básica de programa almacenado 317
 - Unidad de procesamiento central (CPU)* 318
 - Trayectoria de datos simple 318
 - Control de la trayectoria de datos simple 320
3. Implementaciones de la unidad de control 322
 - Unidad de control cableada* 323
 - Memoria e interfaz E/S 325
 - Unidad de control microprogramada* 326
4. Arquitecturas de microprocesador contemporáneas 329
 - Canalización de instrucciones* 329
 - Unidades de hardware en paralelo* 331
 - Jerarquía de memoria* 332
 - Computadora de conjunto complejo de instrucciones (CISC)* 332
 - Computadora de conjunto de instrucciones reducidas (RISC)* 334
5. Arquitecturas de microcontrolador 335
- Resumen y repaso del capítulo 336
- Problemas 336

APÉNDICE MOSFET y transistores de unión bipolar 339

BIBLIOGRAFÍA 343

ÍNDICE 345

Capítulo 1

Representación de números, códigos y conversión de códigos

Tal vez parezca un poco extraño iniciar este libro con un capítulo que corresponde a las representaciones de números y códigos. El procedimiento no es distinto a comenzar con el alfabeto cuando se aprende un lenguaje. Tengamos esto presente hasta que la importancia del orden resulte evidente.

1 SISTEMAS DIGITALES Y ANALÓGICOS

Esta obra aborda el estudio de los llamados *sistemas digitales*. Estos sistemas han ido sustituyendo poco a poco a los más antiguos, denominados *sistemas analógicos*. Los sistemas digitales se encuentran en muchos sectores de la tecnología moderna, siendo el ejemplo más común la *computadora digital*. Los métodos y componentes que se describen en este libro se utilizan también en muchos otros sistemas digitales, por ejemplo:

- Sistemas que controlan las operaciones en un proceso.
 - Señales de tráfico.
 - El flujo de compuestos químicos en una planta química.
 - La temperatura en diversos procesos.
 - La operación de las partes de un motor de automóvil.
 - El seguimiento del tiempo.
- Máquinas que entregan productos (máquinas vendedoras) o que permiten el acceso (puertas de peaje en estacionamientos, estaciones de trenes subterráneos, carreteras de peaje).
- Máquinas electrónicas que graban y reproducen música, voz y video.
- Instrumentación y máquinas médicas.
- El sistema telefónico: teléfonos de tonos por contacto, máquinas contestadoras, centrales de conmutación, sistemas móviles.
- Periféricos de computadora: partes de la pantalla y de la impresora, interfaz del tablero, escáner.
- Control aéreo, procesamiento de señales de radar.
- Instrumentación electrónica: osciloscopio digital, analizador lógico.
- Videojuegos.

Para fines científicos o técnicos, los procesos cuantitativos en el mundo natural, tal como la presión del aire, la velocidad, el voltaje y la posición de un engrane se describen mediante variables. Las relaciones entre variables se expresan en términos de *leyes* o *fórmulas*: las leyes de Newton, la ley de Ohm, la ley de Hooke, entre otras. Las variables se consideran a menudo como *señales*, algunas de las cuales se toman como las causas, o *entradas*, y otras como efectos, o *salidas*, producidas por estas causas mediante la acción de leyes apropiadas. Una fuerza (entrada), por ejemplo, aplicada a un objeto físico origina el movimiento del objeto, con una aceleración (salida) cuyo valor está especificado por la ley de Newton¹.

En la experiencia normal, las señales físicas parecen cambiar de una manera continua. En un día ordinario, la temperatura ambiental aumenta de manera continua hasta alcanzar un máximo, y después disminuye gradualmente hasta alcanzar un mínimo en la noche. Una corriente eléctrica, si bien está compuesta por un flujo de millones de partículas discretas cargadas (usualmente electrones), parece variar de manera continua. Sin embargo, cualquier curva continua se aproxima por medio de una función formada por escalones discretos, como se indica en la figura 1. La aproximación puede hacerse tan precisa como se desee, eligiendo adecuadamente los intervalos en los cuales la variable independiente se descompone. A pesar de que la aproximación de la figura 1 es gráfica, las aproximaciones de este tipo y las soluciones de cualesquiera relaciones matemáticas entre variables pueden obtenerse también por medio de métodos numéricos.

Un proceso en el cual las señales varían de forma continua se conoce como sistema *analógico*; aquél en el cual las señales son discretas se denomina *digital*². Como se indicó antes, las señales continuas pueden hacerse discretas o *digitalizarse* (lo inverso también es posible; resulta factible convertir las señales digitales en una forma analógica³). Todas las variables procesadas en un sistema digital deben ser digitales. De aquí en adelante supondremos que todas las señales a que nos referimos en este libro son digitales; son inherentemente digitales o se han digitalizado⁴.

Un ejemplo en el cual las señales analógicas se digitalizan son las grabaciones de audio digital: discos compactos (CD) y cintas de audio digital (DAT). La forma de onda de la música analógica se muestrea en intervalos de unos cuantos microsegundos, y cada muestra se representa mediante un número digital que varía de 0 hasta el equivalente digital de aproximadamente 4100. Esta información de frecuencia y volumen se almacena en el CD o en el DAT. El reproductor de CD o DAT reconvierte después la información digital en una señal analógica que acciona el altavoz para el disfrute del oyente.

¹ Cualidades como confianza, belleza y justicia no se incluyen evidentemente en esta descripción.

² El término *analógico* utilizado para *continuo* tiene su origen en el uso de un instrumento eléctrico específico o máquina de cómputo, cuya salida es una función continua y cuya estructura interna simula procesos mecánicos, aerodinámicos o de otro tipo por *analogía* entre tales sistemas y los sistemas eléctricos. En consecuencia, el instrumento se denomina *computadora analógica*.

³ Los dispositivos que efectúan estas funciones se conocen como convertidores A/D (analógico-digital) y D/A (digital-analógico), respectivamente.

⁴ Algunas cantidades como el número de estudiantes en una clase, el valor nominal de bonos gubernamentales y el dinero en monedas o billetes, son inherentemente discretas. Considere la máquina expendedora en la que usted compra un refresco o un dulce. Dependiendo de los precios de los productos que incluyen, algunas máquinas sólo aceptan ciertas monedas como entrada. Otras (por ejemplo, aquellas que venden estampillas o pasajes en un sistema de tránsito rápido) únicamente aceptan billetes. La *señal* de dinero en este caso es de manera inherente digital; desde luego, el *producto* comprado también es *digital* puesto que proviene de la máquina en tamaños específicos. Los productos líquidos se almacenan en contenedores de tamaño discreto elegidos arbitrariamente. (Por otro lado, al especificar el tamaño de los recipientes de cerveza permitidos dentro de una jurisdicción gubernamental específica, las autoridades pueden distinguir a los vendedores cuyos productos se entregan en diferentes tamaños, —por ejemplo, la cerveza de microcervecías o en recipientes graduados según un sistema métrico.)

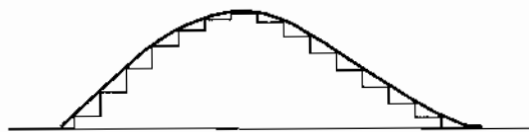


Figura 1. Función continua aproximada mediante un conjunto de valores discretos.

2 HARDWARE, SOFTWARE Y FIRMWARE

Dos amplias categorías definen al campo de la ingeniería de cómputo: *hardware* y *software*. El *hardware* se refiere a los objetos físicos, componentes, circuitos y subensambles acoplados física y electrónicamente para formar sistemas digitales grandes y pequeños. (Como se señaló en la sección anterior, el adjetivo *digital* significa que las señales procesadas por sistemas de este tipo no son señales *continuas*, como aquellas que conforman la música o el habla, o la variación del voltaje del sistema eléctrico de potencia a lo largo del tiempo, sino que son *discretas*, pues sólo toman un número específico de valores.)

En un sentido amplio, el *software* denota las instrucciones que especifican las tareas que efectuará el *hardware*. Suelen distinguirse dos categorías generales de *software*: *sistemas operativos* y *programas de aplicación*. Un sistema operativo de computadora lleva a cabo instrucciones detalladas en cuanto a cómo operar la computadora con el fin de ejecutar su función. Los ejemplos incluyen a DOS (sistema operativo de disco), UNIX y Windows. Un programa de aplicación, por otro lado, especifica las instrucciones que una computadora ejecutará para implementar una tarea específica. Un caso simple consiste en todos los pasos que una computadora debe dar para determinar las raíces de un polinomio. Un caso más general es un programa para realizar una amplia gama de operaciones matemáticas (el programa MATLAB, por ejemplo), un programa procesador de palabras, un juego en el que participan uno o más individuos en una computadora. El *software* —ya sea de sistemas operativos o programas de aplicación— está formado por programas que se instalan en la memoria de una computadora después de la fabricación de ésta. (Normalmente el vendedor de computadoras instala los sistemas operativos; los programas de aplicación casi siempre los instala el usuario.)

Existe también otra categoría, denominada *firmware*. Consiste en el *software* instalado permanentemente en la computadora durante la manufactura. Se dice que está *cableado*. El *firmware*, en consecuencia, es un híbrido entre el *software* y el *hardware*, del cual forma parte integral. Su función estriba en controlar la operación del resto del *hardware*.

Este libro no tiene que ver con el *software* como tal o con el *firmware*; tiene que ver con el *hardware* en un nivel elemental. (Sin embargo, dedicaremos cierto tiempo al lenguaje de descripción de *hardware* denominado ABEL, utilizado en el diseño de *hardware*.) Esto es, aborda principalmente los componentes, circuitos y subensambles que conforman una computadora funcional u otro dispositivo digital. En los capítulos posteriores se analizarán unidades más grandes, cuyo control implica algunos aspectos de programación.

Un vistazo al resto del capítulo y del capítulo 2 revela que la materia del tema contiene una cantidad sustancial de matemáticas. Existen algunos riesgos al iniciar un libro sobre ingeniería de cómputo con temas de matemáticas. Uno de ellos es que muchos de los lectores están interesados en llegar al diseño del *hardware* y quizá sientan impaciencia al verse rodeados de lo que parece no relacionarse con esta meta. Otro riesgo es que algunos lectores han tenido contacto con el tema y podrían pensar que ya lo conocen, por lo que es probable consideren que no vale la pena detenerse en él.

El álgebra booleana expuesta aquí, sin embargo, es esencial para entender la operación de sistemas digitales y, en consecuencia, para diseñarlos.

Además, este material es útil para introducir la terminología y la notación que se utilizarán en el resto del libro. Por consiguiente, incluso si usted *piensa* que conoce el material, se le recomienda estudiarlo. Léalo rápidamente si le es muy familiar; si lo conoce parcialmente, estúdielo

con más detenimiento, dejando tiempo suficiente para efectuar algunos de los ejercicios y problemas. Es posible que necesite repasar este capítulo cuando se haga uso de la representación o códigos numéricos señalados.

A las computadoras algunas veces se les denomina *tritadoras de números* de manera afectiva. Los números que trituran, sin embargo, no son *decimales* sino *binarios*, así como de otros sistemas numéricos relacionados con el binario. Si usted quiere comprender la operación de las computadoras, necesita familiarizarse con la representación numérica. Si ya cuenta con conocimiento acerca del tema, pruébese a sí mismo resolviendo los problemas relevantes al final de este capítulo. Si no dispone de esos fundamentos, será bastante remunerador al dominar el contenido de la siguiente sección. El material en las secciones 3 y 4, acerca de códigos y de conversiones de código, no se necesita hasta el capítulo 4. Si no está familiarizado con el tema, debe estudiar aquellas secciones y resolver los problemas relevantes al final de este capítulo antes de emprender el estudio del capítulo 4.

Los bloques constitutivos de las computadoras y de otros sistemas digitales modernos son los circuitos electrónicos. A pesar de eso, este libro no requiere conocimientos de electrónica. En ocasiones, haremos referencia a ciertos aspectos de los transistores y los circuitos integrados para facilitar la explicación del tema que se estudia, aunque dicha referencia tiene un fin descriptivo y no requiere conocimientos previos. Algunos elementos de transistores y electrónica se cubren en un breve apéndice.

3 SISTEMAS NUMÉRICOS

Como se indicó en la sección 1, los sistemas digitales incluye muchos otros dispositivos además de los preeminentes: las computadoras. La operación de dichos dispositivos podría no requerir la manipulación de números. Las computadoras, sin embargo, procesan bastas cantidades de números y otros datos. Por consiguiente, es importante familiarizarse con aquellos sistemas numéricos por medio de los cuales se expresan los números y con la manera en la cual ejecutan las operaciones aritméticas las computadoras que utilizan estos sistemas. El sistema numérico más familiar en nuestro tiempo es el sistema numérico decimal. Éste utiliza un familiar *alfabeto* de 10 dígitos representados por 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. La representación decimal de un número, como el entero 3756, constituye una notación abreviada para escribir un polinomio en base 10:

$$3756 = 3 \times 10^3 + 7 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$$

El número 10 es la *base* del sistema decimal. En la forma usual de expresar el entero 3756, las potencias de 10 se encuentran sugeridas por la posición que ocupan los dígitos, empezando con la potencia 0 a la derecha y procedente en orden ascendente hacia la izquierda. De ese modo, el sistema decimal es un *sistema numérico de posición*.

Cualquier sistema numérico de posición puede crearse especificando dos cosas: la base b y los b dígitos que integran el alfabeto. Cuando b es 10 o menos, es usual utilizar los dígitos 0 a $b - 1$ para el alfabeto. Si b es mayor que 10, se necesitan nuevos símbolos para representar los dígitos adicionales. Indicamos que un número N se expresa en base b con $(N)_b$. Si los dígitos en el alfabeto se denominan a_i , entonces, por analogía con en el sistema decimal, N puede escribirse como un polinomio en b de la manera siguiente:

$$(N)_b = a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \dots + a_1b^1 + a_0b^0 + a_{-1}b^{-1} + \dots + a_{-m}b^{-m} \quad (1)$$

El número dado tiene una parte *entera*, $a_{n-1}a_{n-2}\dots a_1a_0$, y una parte *fraccionaria*, $a_{-1}a_{-2}\dots a_{-m}$.

El dígito a_{n-1} que multiplica la potencia más alta de b es el *dígito más significativo*; el dígito a_{-m} que multiplica a la potencia más baja de b es el *dígito menos significativo*⁵.

⁵ Por qué *msb* y *lsb* y no *msd* y *lsd*, se explicará en breve.

El sistema binario y otros sistemas numéricos

El sistema numérico de posición de mayor importancia en las computadoras digitales es el *binario*. Su base es 2, y su alfabeto está compuesto por 2 dígitos: 0 y 1. Al igual que un número decimal, un número binario se escribe como una secuencia yuxtapuesta de dígitos, binarios en este caso —por ejemplo, 1101001. De acuerdo con 1), ésta es una forma corta correspondiente a un polinomio en 2.

Cada dígito en el alfabeto binario transmite un mínimo de información, como por ejemplo, si un interruptor está *activado* o *desactivado*. Esta unidad de información se denomina *bit*. Por esta razón, se acostumbra nombrar a un dígito binario bit. ("Un bit" también puede considerarse como una abreviatura de *binary digit* (dígito binario).) Así, *msb* y *lsb* son abreviaturas para *bit más (menos) significativo*.

Otros sistemas numéricos que tienen aplicaciones útiles son:

Base 8: sistema *octal*,

Base 16: sistema *hexadecimal*.

Puesto que el último sistema requiere un alfabeto con más de 10 dígitos, es costumbre utilizar los 10 dígitos decimales más las primeras seis letras (mayúsculas) del alfabeto inglés. La figura 2 muestra la representación de los enteros decimales de 0 a 15 en diversos sistemas numéricos. Estudie la tabla y familiarícese con los patrones de dígitos en las últimas tres columnas.

Existe cierta falta de consistencia en esta tabla. En todas menos en la columna decimal, cada número tiene el mismo número de dígitos. Para ser consistentes, debemos agregar el dígito 0 delante de los primeros 10 números decimales, esto es, en la posición de las decenas. Pero esto no suele hacerse en el caso de los números decimales, por lo cual lo evitaremos. Si lo hubiéramos hecho de esta manera, como en el caso de los números hexadecimales, entonces cada uno de los primeros ocho números de la tabla se habría representado por medio de la misma secuencia de dígitos en todos los casos salvo el sistema binario. Una segunda observación es que la misma secuencia de dígitos puede representar números diferentes dependiendo de la base del sistema numérico. De esta manera, la secuencia 12 representa al decimal doce, al octal diez y al hexadecimal dieciocho. (Usted puede ampliar la tabla para confirmar esto.)

Sistema numérico			
Decimal (Base 10)	Binario (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Figura 2. Representaciones de números enteros.

Posiblemente le sorprenda toda esta concentración de diferentes sistemas numéricos; ¿por qué no usar sólo el sistema decimal en la computación, o por qué considerar otros sistemas numéricos si las computadoras de la actualidad utilizan el sistema binario? Los primeros dispositivos físicos que se utilizaron para efectuar los cálculos modernos eran interruptores y relevadores⁶. Un interruptor tiene inherentemente dos posiciones: activado y desactivado. Estas posiciones pueden utilizarse fácilmente para representar los dígitos en un sistema binario. Los dispositivos electrónicos posteriores cuentan también con dos posiciones estables que se distinguen con facilidad. Si se recurriera al sistema decimal para efectuar la aritmética de la computadora, resultarían necesarios dispositivos físicos con 10 posiciones estables para representar los 10 dígitos. A pesar de que es concebible diseñar componentes electrónicos con 10 posiciones distintas, resultarían probablemente bastante costosos. Además, los sistemas que utilizan tales componentes no serían confiables, pues resultaría difícil asegurar la estabilidad de las 10 posiciones.

De acuerdo; el caso del sistema binario *contra* el decimal se ha aclarado; pero, usted podría preguntarse, ¿qué ocurre con los sistemas octal y hexadecimal? Una mirada a la figura 2 indica que incluso para estos enteros de valor reducido, el número de dígitos binarios necesarios para representar un número es bastante grande. (Un número decimal de 6 dígitos requeriría 18 dígitos binarios.) Un número representado en el sistema octal necesita alrededor de tres veces menos dígitos que el mismo número representado en el sistema binario, y un sistema hexadecimal tiene aproximadamente una ventaja de 4 a 1. La siguiente sección explica lo fácil que es convertir del sistema binario a uno de estos sistemas, y viceversa. De esta manera, aun cuando en las computadoras no se utilicen dispositivos físicos con 8 o 16 posiciones estables, la conveniencia de un mínimo de dígitos en la representación numérica y la fácil conversión al sistema binario hacen que los sistemas octal y hexadecimales sean motivo de un breve estudio.

Conversiones de base

Un problema frecuente consiste en convertir un número expresado en una base particular en otro expresado en una base diferente. En virtud de nuestra familiaridad con la aritmética decimal, se distinguen dos casos:

- Convertir a partir de algún otro sistema numérico al sistema decimal.
- Convertir a partir del sistema decimal a uno con otra base.

Conversión al sistema decimal

Suponga, por ejemplo, que un número binario se va a convertir en un número decimal. Esto se efectúa fácilmente formando, de acuerdo con 1), el polinomio en 2 correspondiente al número binario y expresando después cada potencia de 2 como un número decimal. Como ejemplo,

$$\begin{aligned}(101011.11)_2 &= 2^5 + 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-2} \\ &= 32 + 8 + 2 + 1 + 0.5 + 0.25 = 43.75\end{aligned}$$

Advierta que en esta conversión se aplica la aritmética decimal.

Ejercicio 1. En un número decimal, el punto que separa la parte entera y la parte fraccionaria se denomina *punto decimal*. ¿Cuál sería un nombre razonable para el punto que separa las dos partes de un número binario, como en el ejemplo precedente? ¿Cuál sería la situación del punto correspondiente en los sistemas octal y hexadecimal? •

⁶ Shannon Claude E.: "A Symbolic Analysis of Relay and Switching Circuits," *Trans AIEE*, 57, 1938, 713-723. Éste es un artículo bastante fácil de leer; nadie que no lo haya leído puede afirmar que es letrado en cómputo.

Ejercicio 2. El siguiente es un número octal: 2034. Utilizando el ejemplo previo de conversión del sistema binario al decimal como modelo, describa cómo convertir un número octal en un número decimal. Convierta después el número octal 2034 en decimal (hágalo primero y luego consulte la respuesta).

Respuesta⁷

Conversión a partir del sistema decimal

El proceso inverso, la conversión del sistema decimal a otro sistema numérico, es un poco más complicado. Vamos a recurrir a la conversión del sistema decimal al binario, como modelo para cualquier conversión. Utilizaremos de nuevo la aritmética decimal. Las partes entera y fraccionaria se manejan por separado.

El entero decimal N se va a convertir en un número binario. (La especificación de la base 10 se ha omitido por conveniencia.) Advierta en 1), con $b = 2$, que el último dígito en la parte entera es 1 si el número es impar y 0 si éste es par. Dejando el último dígito de lado, es posible factorizar un 2 en la resta. El número que queda después de que el 2 se factoriza puede tratarse exactamente de la misma manera; este último dígito será 1 sólo si ese número es impar, y es posible factorizar un 2 en la parte restante. A continuación mostramos el aspecto de la expresión que se obtiene después de estos dos pasos:

$$\begin{aligned} N &= a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \dots + 2a_1 + a_0 \\ &= 2(a_{n-1}2^{n-2} + a_{n-2}2^{n-3} + \dots + 2a_2 + a_1) + a_0 \\ &= 2[2(a_{n-1}2^{n-3} + a_{n-2}2^{n-4} + \dots + 2a_3 + a_2) + a_1] + a_0 \end{aligned} \quad (2)$$

Este procedimiento detallado, puede prolongarse hasta que sólo quede un dígito dentro del paréntesis interno.

Por tanto, empezamos con un número decimal N y procedemos en sentido opuesto a partir del desarrollo precedente. Si se divide entre 2 el número decimal dado se obtiene un cociente Q_0 y un residuo R_0 . Por consiguiente, el número puede escribirse

$$N = 2Q_0 + R_0$$

El residuo R_0 es 1 si N es impar, y 0 si N es par. Posteriormente el cociente Q_0 puede analizarse como el número original N . Si se divide Q_0 entre 2, se obtiene un nuevo cociente Q_1 y un nuevo residuo R_1 . El último es 1 o 0, dependiendo de si el primer cociente Q_0 es par o impar. De nuevo $Q_0 = 2Q_1 + R_1$. El número original puede escribirse ahora

$$N = 2(2Q_1 + R_1) + R_0$$

No es difícil observar el patrón cuando cada cociente se divide entre 2. Advierta que en algún punto en el proceso de dividir entre 2, el cociente se volverá 3 o 2. Una división adicional entre 2 producirá un cociente igual a 1; ya no es posible una división entre 2, y el proceso finaliza. El resultado del paso posterior al inmediato anterior y el resultado final después de k divisiones entre 2 son

$$\begin{aligned} N &= 2[2(2Q_2 + R_2) + R_1] + R_0 = 2^3Q_2 + 2^2R_2 + 2R_1 + R_0 \\ N &= 2^k + R_{k-1}2^{k-1} + R_{k-2}2^{k-2} + \dots + R_12^1 + R_0 \end{aligned} \quad (3)$$

⁷ La base es 8. Cada dígito de un número octal se multiplica por 8 elevado a una potencia basada en la posición del dígito:

$$(2034)_8 = 2 \times 8^3 + 0 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 1024 + 24 + 4 = 1052$$

Ejercicio 2. El siguiente es un número octal: 2034. Utilizando el ejemplo previo de conversión del sistema binario al decimal como modelo, describa cómo convertir un número octal en un número decimal. Convierta después el número octal 2034 en decimal (hágalo primero y luego consulte la respuesta).

Respuesta⁷

Conversión a partir del sistema decimal

El proceso inverso, la conversión del sistema decimal a otro sistema numérico, es un poco más complicado. Vamos a recurrir a la conversión del sistema decimal al binario, como modelo para cualquier conversión. Utilizaremos de nuevo la aritmética decimal. Las partes entera y fraccionaria se manejan por separado.

El entero decimal N se va a convertir en un número binario. (La especificación de la base 10 se ha omitido por conveniencia.) Advierta en 1), con $b = 2$, que el último dígito en la parte entera es 1 si el número es impar y 0 si éste es par. Dejando el último dígito de lado, es posible factorizar un 2 en la resta. El número que queda después de que el 2 se factoriza puede tratarse exactamente de la misma manera; este último dígito será 1 sólo si ese número es impar, y es posible factorizar un 2 en la parte restante. A continuación mostramos el aspecto de la expresión que se obtiene después de estos dos pasos:

$$\begin{aligned} N &= a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + 2a_1 + a_0 \\ &= 2(a_{n-1}2^{n-2} + a_{n-2}2^{n-3} + \cdots + 2a_2 + a_1) + a_0 \\ &= 2[2(a_{n-1}2^{n-3} + a_{n-2}2^{n-4} + \cdots + 2a_3 + a_2) + a_1] + a_0 \end{aligned} \quad (2)$$

Este procedimiento detallado, puede prolongarse hasta que sólo quede un dígito dentro del paréntesis interno.

Por tanto, empezamos con un número decimal N y procedemos en sentido opuesto a partir del desarrollo precedente. Si se divide entre 2 el número decimal dado se obtiene un cociente Q_0 y un residuo R_0 . Por consiguiente, el número puede escribirse

$$N = 2Q_0 + R_0$$

El residuo R_0 es 1 si N es impar, y 0 si N es par. Posteriormente el cociente Q_0 puede analizarse como el número original N . Si se divide Q_0 entre 2, se obtiene un nuevo cociente Q_1 y un nuevo residuo R_1 . El último es 1 o 0, dependiendo de si el primer cociente Q_0 es par o impar. De nuevo $Q_0 = 2Q_1 + R_1$. El número original puede escribirse ahora

$$N = 2(2Q_1 + R_1) + R_0$$

No es difícil observar el patrón cuando cada cociente se divide entre 2. Advierta que en algún punto en el proceso de dividir entre 2, el cociente se volverá 3 o 2. Una división adicional entre 2 producirá un cociente igual a 1; ya no es posible una división entre 2, y el proceso finaliza. El resultado del paso posterior al inmediato anterior y el resultado final después de k divisiones entre 2 son

$$\begin{aligned} N &= 2[2(2Q_2 + R_2) + R_1] + R_0 = 2^3Q_2 + 2^2R_2 + 2R_1 + R_0 \\ N &= 2^k + R_{k-1}2^{k-1} + R_{k-2}2^{k-2} + \cdots + R_12^1 + R_0 \end{aligned} \quad (3)$$

⁷ La base es 8. Cada dígito de un número octal se multiplica por 8 elevado a una potencia basada en la posición del dígito:

$$(2034)_8 = 2 \times 8^3 + 0 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 1024 + 24 + 4 = 1052$$

De este modo, la representación binaria de un entero decimal distinto de 0 es $1 R_{k-1} R_{k-2} \dots R_1 R_0$.

El proceso se ilustra en la tabla siguiente, que muestra la conversión del decimal 234 a la forma binaria.

	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	
Q_i	117	58	29	14	7	3	1	—
R_i	0	1	0	1	0	1	1	1
	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7

La conversión a partir de decimales de la parte fraccionaria de un número al sistema binario sigue un procedimiento similar, excepto cuando una fracción decimal determinada F_0 se *multiplifica* primero por 2 en vez de dividirse entre 2. El número que se obtiene tendrá una parte entera B_1 (1 si la fracción original es ≥ 0.5 y 0 en cualquier otro caso) y una parte fraccionaria F_1 . Estos pasos se repiten con F_1 , y el proceso continúa, lo cual da origen a la serie localizada en la primera columna de la tabla siguiente.

$F_0 = 0.40625$	$2F_{i-1}$	B_i	F_i
$2F_0 = B_1 + F_1$	0.8125	0	0.8125
$2F_1 = B_2 + F_2$	1.625	1	0.625
$2F_2 = B_3 + F_3$	1.25	1	0.25
	0.5	0	0.5
$2F_i = B_{i+1} + F_{i+1}$	1.0	1	0.0

(En la columna izquierda, no figura B_0 .) El proceso termina si algún residuo fraccionario F_{i+1} se vuelve 0. La fracción binaria será $0.B_1 B_2 B_3 \dots B_{i-1}$. El trabajo distribuye en una tabla, a la derecha de la anterior, para la fracción decimal 0.40625 convertida a la forma binaria.

Es posible que el proceso no termine debido a que el residuo fraccionario nunca se convierte en 0. En tal caso, la forma binaria de la fracción no tiene un número finito de dígitos.

Del octal o hexadecimal al binario

La conversión de números a otras bases sigue uno de los patrones descritos. Sin embargo, existe un procedimiento más simple en los casos especiales de conversión del sistema octal al binario y del hexadecimal al binario. La razón radica en que las bases de los sistemas octal y hexadecimal pueden expresarse en el número 2 elevado a alguna potencia: 2^3 y 2^4 , respectivamente.

Por consiguiente, para convertir un número binario a la forma octal, cada dígito octal se representa mediante tres dígitos binarios. En consecuencia, un número binario dado se divide en conjuntos de tres dígitos empezando en el punto binario, y procediendo hacia la izquierda en el caso de la parte entera del número y hacia la derecha en el caso de la parte fraccionaria. Cada conjunto de tres dígitos binarios se sustituye entonces mediante el dígito octal correspondiente. Por ejemplo

$$\begin{array}{ccccccc} (10 & 110 & 111 & 011.101 & 1)_2 & = & (2673.54)_8 \\ 2 & 6 & 7 & 3 & \cdot & 5 & 4 \end{array}$$

(Los espacios entre los conjuntos binarios de tres dígitos no se encuentran normalmente ahí; tienen que proporcionarse para facilitar la visualización.) En la operación inversa, la conversión de un número octal determinado a la forma binaria, cada dígito octal se sustituye por el conjunto correspondiente de tres dígitos binarios. Así,

$$(356.07)_8 = (011 \ 101 \ 110.000 \ 111)_2$$

Ejercicio 3. Por analogía con las conversiones precedentes del sistema binario a la forma octal y viceversa, describa la forma de convertir un número binario a la forma hexadecimal. Repita la conversión de un número hexadecimal a la forma binaria. Utilizando su esquema, efectúe las siguientes conversiones:

$$(11001011101.011)_2 = (x)_{16}$$

$$(3C9.E)_{16} = (y)_2$$

Respuesta⁸**Aritmética binaria**

Una computadora digital consume la mayor parte de su vida útil a la realización de operaciones de aritmética binaria. Por tanto, entender el funcionamiento básico de las computadoras requiere conocer la aritmética binaria. Sin embargo, las computadoras digitales tienen limitaciones. Comprender el funcionamiento de la computadora significa también conocer estas limitaciones y los procedimientos para evitarlas. Esta sección le será de utilidad en este aspecto.

Las reglas de la aritmética binaria son similares a las de la aritmética decimal. Los conceptos de *acarreo* y *préstamo*, por ejemplo, se aplican también a la aritmética binaria.

Suma

Si la suma decimal de dos dígitos es igual o mayor que la base, 10, se origina un acarreo de 1. En el caso de los números binarios, la suma de dos dígitos binarios puede, a lo más, *ser igual* a la base. El único acarreo distinto de cero que *puede* darse es 1. La segunda columna en la figura 3 muestra el resultado de sumar cualquier combinación de dos dígitos binarios. El único acarreo distinto de cero se genera cuando se suman los dígitos 1 y 1.

Como en la suma decimal, dos números binarios de dígitos múltiples se suman empezando con los bits menos significativos. Cualquier acarreo que resulte se añade a la suma de los siguientes bits significativos a la izquierda.

La siguiente tabla ilustra la suma de dos números binarios.⁹ La línea superior representa los acarreos generados.

	Suma binaria	Equivalente decimal
Acarreo:	10111.1	
	1011.11	11.75
	+ 1001.01	+ 9.25
	10101.00	21.00

Note que, debido a que el acarreo se genera al sumar los bits más significativos, la suma binaria tiene 5 bits en su parte entera, aunque la parte entera de cada operando cuenta únicamente con 4 bits. Este acarreo recibe el nombre de *acarreo fuera* del bit más significativo.

Dígitos	Suma		Resta		Multiplicación
$a_1 a_2$	$a_1 + a_2$		$a_1 - a_2$		$a_1 \cdot a_2$
	Suma	Acarreo	Diferencia	Préstamo	Producto
0 0	0	0	0	0	0
0 1	1	0	1	1	0
1 0	1	0	1	0	0
1 1	0	1	0	0	1

Figura 3. Operaciones de aritmética binaria.

⁸ $x = 65D.6$, $y = 001111001001.1110$

⁹ En todos los ejemplos que siguen, usted debe efectuar las operaciones indicadas independientemente y confirmar de esa manera los resultados que se indican.

Resta

En la resta decimal, si el dígito que se va a restar es mayor que el dígito correspondiente en el minuendo, se *toma* un uno del primer dígito distinto de cero a su izquierda. El concepto de tomar prestado se aplica también en la sustracción binaria. Así, siempre que un número binario 1 se resta de un 0 en alguna posición, debe pedirse prestado un 1 al primer dígito distinto de 0 a la izquierda de esta posición. Esto dejará un 0 en esa posición y cambiará en 1 todos los dígitos 0 que intervienen hasta la posición en cuestión. El 0 en esa posición se convierte en 10 (decimal 2); la resta del 1 deja un dígito 1. El proceso se ilustra mediante el siguiente ejemplo. Los dos renglones superiores corresponden a los dígitos que se piden prestados en una posición determinada y a los cambios realizados en los dígitos a la derecha de la posición a partir de la cual se pide prestado un 1.

Sustracción binaria	Detalles	Equivalente decimal
	11 1 : Prestamo	
	01 0 : Dígitos cambiados	
10011.01	10011.01	19.25
-01100.11	<u>-01100.11</u>	<u>-12.75</u>
	00110.10	6.50

(Puede verificar la respuesta sumando el sustraendo; el resultado debe ser el minuendo. ¿Es así?) El caso en el que un número se resta de un número menor se explica en la siguiente sección.

Multiplicación

La multiplicación de dos números binarios se efectúa al multiplicar el multiplicando por cada dígito del multiplicador y al sumar estos productos parciales, con la posición de cada producto parcial sucesivo corrida una unidad a la izquierda. Si un dígito del multiplicador es 0, todos los dígitos del producto parcial correspondiente son 0; si un dígito del multiplicador es 1, el producto parcial correspondiente es el mismo que el multiplicando. A continuación se ofrece un ejemplo.

Multiplicación binaria	Equivalente decimal
10011	19
$\times 101$	$\times 5$
<u>10011</u>	<u>95</u>
00000	
<u>10011</u>	
1011111	

División

Los pasos que se efectúan en la división de dos números binarios son bastante similares a los de la división decimal, pero con algunas diferencias. En cada paso, el divisor de n bits se aplica a los n bits más altos del dividendo, que empieza con un 1, ya sea una vez o cero veces. En el último caso, el bit correspondiente del cociente parcial es cero. Luego se añade el siguiente bit más significativo del dividendo, y el proceso continúa hasta que aparece 1 en el cociente parcial. Después se sustrae el divisor del dividendo en ese punto y el proceso continúa. Consideremos un ejemplo.

$$\begin{array}{rcl}
 \text{Dividendo} \rightarrow & 100101011 & \underline{10111} \leftarrow \text{Divisor} \\
 & \underline{10111} & 01101 \leftarrow \text{Cociente} \\
 & 011100 & \\
 & \underline{10111} & \\
 & 010111 & \\
 & \underline{010111} & \\
 & 00000 & \leftarrow \text{Residuo}
 \end{array}$$

Complementos: a dos y a uno

Las sumas y multiplicaciones repetidas de números binarios generarán resultados en los cuales el número de dígitos puede crecer indefinidamente. Sin embargo, las "circuits integrados" que forman el corazón de la unidad de procesamiento en una computadora digital sólo pueden manipular números con una cantidad fija de dígitos. Esta cantidad recibe el nombre de *longitud de palabra*. Las primeras computadoras tenían longitudes de palabra de 8. En los últimos tiempos, son comunes las computadoras con longitudes de palabra de 64 bits.

Si la longitud del número que resulta de una operación aritmética sobre números binarios excede la longitud de palabra n de la computadora, se retiene un número compuesto de los n bits menos significativos; los bits más significativos constituirán un desbordamiento. Existe un mecanismo matemático para tratar este desbordamiento, aunque no consideraremos este tema aquí.

Los números sobre los cuales opera una computadora digital pueden ser positivos o negativos. Los números negativos surgen cuando se sustrae un número mayor de uno menor. Resultaría en extremo valioso que hubiera un mecanismo que permitiera a la computadora seguir un procedimiento sencillo sin que importe que la operación que se va a efectuar sea una suma o una resta, y sin que tenga relevancia el hecho de que los números sean positivos o negativos. Debe existir una manera única de determinar a partir del resultado de la operación si el número resultante es positivo o negativo. Ese mecanismo es el tema de esta sección.

Cuando un dígito, aparte de 0, se sustrae de la base del sistema numérico en el cual se expresa, el resultado es el *complemento con respecto a la base*. En el sistema decimal, éste es el complemento con respecto a 10, o el *complemento a diez* (o complemento a 10). El complemento a diez de 3 es 7, por ejemplo. En general, el complemento a diez de un entero decimal se obtiene restándolo de 10 elevado a una potencia igual al número de dígitos del entero. Así, el complemento a diez de 461 es $10^3 - 461 = 539$.

El mismo concepto puede aplicarse a los números binarios. Sea N un entero binario con n dígitos. El *complemento a dos* (o complemento a 2) de N , que se denota N_{2c} , se obtiene al restar el número de 2^n expresado en forma binaria, esto es, restando el número binario de 1 seguido por n ceros:

$$N_{2c} = 2^n - N. \quad \text{Todo expresado en el sistema binario} \quad (4)$$

Para ejemplificar el complemento a dos, sea $N = 110010$; $n = 6$. Así, $N_{2c} = 1000000 - 110010 = 001110$ (compruebe este resultado).

Una manera más simple de obtener el complemento a dos de un número N puede generalizarse a partir del ejemplo:

- Se retiene el 1 menos significativo del número.
- También se retienen los ceros menos significativos que este 1.
- Todos los demás dígitos se sustituyen por sus complementos: 0 por 1 y 1 por 0.

Las diferentes maneras de manejar el 1 menos significativo y los demás dígitos es el resultado de "pedir prestado" cuando se efectúa la sustracción. Únicamente el msb en $(2^n)_2$ es un 1; cuando éste se pide prestado, todos los ceros subsecuentes hasta la posición del 1 menos signi-

ficativo del número se convierten en 1. Luego, al sustraer un 1 de un 1, queda un 0; la sustracción de un 0 de un 1 deja un 1. De ese modo, la sustracción de cada dígito en N de 1 equivale a sustituir cada dígito por su complemento, salvo en el caso del 1 menos significativo. En este caso, se sustrae 1 de 10 (binario), y queda un 1.

Incluso esta diferencia entre el 1 menos significativo y los otros desaparecería si restamos 1 del resultado. Esto es, dado un entero binario N con n dígitos, se forma el complemento a dos: $N_{2c} = (2^n)_2 - N$. Después de esto se resta 1:

$$N_{1c} = N_{2c} - 1 = (2^n)_2 - 1 - N \quad (5)$$

Esto se conoce como el *complemento a uno* (o complemento a 1), N_{1c} . El complemento a uno de 1011010 es

$$10000000 - 1 - 1011010 = 1111111 - 1011010 = 0100101$$

Por consiguiente, el complemento a uno de un entero binario es justamente ese entero binario con cada bit 1 sustituido por un 0 y cada bit 0 reemplazado por un 1. ¡Qué mayor sencillez puede lograrse!¹⁰

Dado un entero binario N , los dos complementos de N pueden obtenerse a partir de ese entero de dos maneras: ya sea directamente, como se describió antes, o determinando primero el complemento a uno (intercambiando trivialmente ceros y unos, como se acaba de describir) y agregando luego un 1 al bit menos significativo.

Ejercicio 4. Encuentre directamente el complemento a dos de los siguientes números binarios y confirme cada respuesta determinando el complemento a uno: a) 11111, b) 10000, c) 0001, d) 00100.

Respuesta¹¹

(Este ejercicio requiere solamente la ejecución de un algoritmo específico. Sin embargo, si todo lo que usted hace es realizar el trabajo, marcar la *respuesta* y continuar, perderá oportunidades importantes de observar y aprender. Note en b), por ejemplo, un número cuyo complemento a dos es idéntico a él. ¿Puede usted pensar en otros casos que tengan esta propiedad? ¿Puede detectar otros casos en el ejercicio en los que un número y su complemento a dos tengan alguna propiedad interesante?)

El complemento a dos es otro sistema para representar números. Verifique que éste, además, es un sistema numérico de posición.

Aunque no es evidente a partir del desarrollo anterior, el complemento a dos proporciona una manera de manejar números negativos. Se adoptará la siguiente representación mezclada de números.

Un número no negativo (número positivo o cero) se representará mediante su forma binaria; un número negativo se representará por medio del complemento a dos de su magnitud.

Esta forma mezclada se conoce como *representación numérica de complemento a dos*.

Dado una serie de dígitos binarios, ¿cómo indicamos si representa un número positivo o uno negativo? Una clave proviene del hecho de que se trata de una decisión indistinta, o binaria. Si dejamos que *positivo* se represente mediante uno de los dos dígitos binarios y *negativo* por el otro, podríamos dejar una posición particular en la secuencia binaria que represente el signo del

¹⁰ El mismo concepto en el sistema decimal es el *complemento a nueve*; inténtelo en un número decimal.

¹¹ a) 00001, b) 10000, c) 1111, d) 11100.

número. La única posición que puede utilizarse para este propósito es la del msb, ya que ésta es la única posición cuyo dígito puede controlarse, como lo explicaremos a continuación.

Es posible asegurar que el msb siempre será el mismo para un número positivo si la longitud de palabra es lo suficientemente larga. En ese caso, el msb para un número positivo debe ser 0. Suponga que la longitud de palabra es 6. El número positivo mayor que es posible representar en el sistema considerado es 011111, o el decimal 31. Si se esperan números mayores, entonces, para un número positivo, se necesita una longitud de palabra mayor para conseguir que el msb sea 0.

Ejercicio 5. Determine el entero decimal más grande (positivo) que puede indicarse en la representación numérica de complemento a dos para la longitud de palabra común actual de 64 bits. ♦

Ejercicio 6. Supongamos que A es un entero binario positivo de n bits. Supongamos, asimismo, que la longitud de palabra es de al menos 2^{n+1} . Demuestre que el msb de la representación numérica de complemento a dos será 0 para A y 1 para $-A$. ♦

Ejercicio 7. El número de enteros negativos y no negativos distintos de n bits, que puede indicarse mediante una representación numérica de complemento a dos, se denomina su *rango*. Determine el rango en términos de n y especifique cuántos enteros son negativos. ♦

Suma de números binarios

Ya contamos con un fundamento para el estudio de la suma o resta de dos enteros de n bits $\pm A$ y $\pm B$, donde A y B son no negativos. Recuerde que se va a utilizar la representación numérica de complemento a dos; esto es, los números no negativos se representarán en números binarios y los negativos mediante el complemento a dos de la magnitud correspondiente. Suponemos que la longitud de palabra de la máquina que efectuará estas operaciones es lo suficientemente larga de manera que el msb de A y de B será 0.

Planeamos obtener la suma o diferencia de dos enteros, cada una de las cuales puede ser positiva o negativa. Es posible escribir el caso más general como $(\pm A) \pm (\pm B)$. Hay un total de ocho combinaciones de signos $+$ y $-$. Sin embargo, puesto que la resta es equivalente a la suma del negativo del sustraendo, estos casos no son del todo distintos; así, $A - (-B)$ es lo mismo que $A + B$. Por ello sólo hay en realidad tres casos distintos, y éstos implican únicamente la suma (confirme esta conclusión analizando todas las posibilidades.)

Los tres casos distintos que deben considerarse son:

Caso 1: $A + B$. La suma de dos enteros no negativos.

Caso 2: $-A + (-B)$. La suma de dos enteros negativos.

Caso 3: $A + (-B)$. La suma de un entero positivo y uno negativo.

Estos tres casos se explicarán por separado.

Caso 1 $A + B$. La suma de dos enteros positivos de n bits. Aun cuando A y B pueden encontrarse dentro del rango, su suma puede excederlo. En dicho caso habrá un desbordamiento, que es posible detectar al observar el acarreo en el msb. Puesto que ninguno de los enteros que se va a sumar es negativo, los dos bits más significativos son 0. Por consiguiente, el *acarreo* de salida del msb debe ser 0. Sin embargo, si la suma no está dentro del rango, entonces el *acarreo* de entrada en la posición del bms (el cual se convierte en el msb) será 1. Así, el desbordamiento se detecta de la manera siguiente.

Si el acarreo de entrada en el mbs es el mismo que el acarreo de salida (ambos 0), entonces no hay desbordamiento; si el acarreo de salida (0) es diferente del acarreo de entrada (1), entonces

ces hay un desbordamiento. En este caso, sencillamente advertimos el hecho y lo descartamos. A continuación se presentan dos ejemplos para $n = 6$; el rango en este caso corresponde a $2^{6-1} - 1 = 31$.

	00111	Acarreo	01111
14	001110	14	001110
+ 11	+ 001011	+ 22	+ 010110
25	011001	36	100100

En el caso de $14 + 11 = 25$, la suma está en el rango; por tanto, no debe haber desbordamiento. Esto se confirma por el hecho de que el acarreo (subrayado) de entrada y el acarreo de salida de la posición del bms son iguales: ambos 0. El bit de signo de la suma es 0, lo que indica un número positivo, como debe ser. En el segundo caso, la suma no está en rango, y la presencia de un desbordamiento se indica por los acarreos en la posición del bms: estos son diferentes. El bit de signo muestra erróneamente que la suma será negativa. Pero como se ha detectado el desbordamiento, esto se ignora.

Caso 2 $-A + -(B)$. La suma de dos números negativos. Puesto que ambos números son negativos, ambos se representarán en complemento a dos. Por consiguiente,

$$(2^n - A) + (2^n - B) = 2^n + [2^n - (A + B)]$$

Puesto que se supone que tanto $-A$ como $-B$ están en rango, cada una de las representaciones de complemento a dos tendrá un bit de signo 1. Por consiguiente, cuando éstos se suman, el acarreo de salida del msb debe ser 1. En vista de que la suma de los dos números negativos debe ser negativa, el msb de la suma también debe ser 1. Lo anterior ocurrirá únicamente si el acarreo dentro de la posición del msb es también 1 —igual al acarreo de salida del msb. De nuevo, si el acarreo de entrada (0) y el acarreo de salida (1) del msb no son iguales, ocurrirá un desbordamiento. Note que el primer 2^n a la derecha en la expresión precedente representa el acarreo de salida del msb. A continuación se presentan dos ejemplos para $n = 6$.

	Decimal	Binario	Complemento a dos	
			1111	Acarreo
	(-18)	(-010010)	101110	
	+ (-11)	+ (-001011)	+ 110101	
Suma	(-29)	(-011101)	100011	

	Decimal	Binario	Complemento a dos	
			1011	Acarreo
	(-18)	(-010010)	101110	
	+ (-27)	+ (-011011)	+ 100101	
	(-45)	(-101101)	010011	

En el primer caso, la suma (-29) está dentro del rango -32 a 31 . Cuando se suman los complementos a dos de los dos números, el acarreo de entrada y el acarreo de salida del msb corresponden ambos a 1, lo que implica que no hay desbordamiento. El resultado de la suma es el complemento a dos de -29 , la respuesta correcta.

En el segundo caso, la suma (-45) está fuera de rango, por lo que esperamos un desbordamiento. Éste se detecta por el hecho de que el acarreo de entrada en la posición del msb es diferente del acarreo de salida. Observe que el resultado de 6 bits correspondiente a tomar la suma en forma errónea tiene 0 como msb, lo cual designa normalmente un número positivo. Pero como se ha detectado un desbordamiento, interpretamos correctamente este número como negativo.

Caso 3 $A + (-B)$. La suma de un entero positivo y de uno negativo.

$$A + 2^n - B = 2^n + (A - B) \quad \text{para } A \geq B \quad (a)$$

$$= 2^n - (B - A) \quad \text{para } A < B \quad (b)$$

Ejercicio 8. Analice el siguiente caso. A partir de los resultados de la representación numérica de complemento a dos, determine de qué modo será posible indicar si la diferencia estará en rango, si habrá un desbordamiento, y si el acarreo de entrada y de salida de la posición del msb son iguales en los dos casos. Proponga un ejemplo para $n = 6$ y valores numéricos correspondientes a $25 - 17$ y $25 - 30$. (La respuesta es demasiado larga para ponerla en un pie de página; se presenta a continuación. No la consulte antes de realizar el ejercicio.)

Respuesta. Si tanto A como B están en rango, entonces la diferencia estará en rango en ambos casos. Para $A \geq B$, el 2^n en $a)$ muestra que habrá un acarreo igual a 1 de salida de la posición del msb, el cual se ignora. Para $A < B$, la diferencia será negativa y estará en rango, por lo que no habrá acarreo de salida de la posición del msb. Puesto que uno de los bits de signo es 1 y el otro 0, y puesto que el acarreo de salida del msb es 0, el acarreo de entrada del msb debe ser también 0. Esto da lugar a que el msb en la diferencia sea 1, lo que significa que la diferencia es negativa. Ejemplos:

	11111	<i>Acarreo</i>		
25	011001		25	011001
$+(-17)$	$+ 101111$		$+ (-30)$	<u>100010</u>
8	001000		-5	111011

En ambos casos, el acarreo de entrada de la posición bms es el mismo que el acarreo de salida (1 y 0, respectivamente). Por consiguiente, el resultado está en rango y no hay desbordamiento, como se esperaba. Los bits de signo muestran una diferencia positiva en el primer caso y una negativa en el segundo. El último resultado es el complemento a dos de -5 . ♦

La razón para considerar sólo la suma y de abordar la resta como un caso especial radica en que realmente las computadoras lo hacen de esa manera. Aunque es posible construir un sustractor en el hardware, se acostumbra utilizar en las computadoras sólo sumadores para efectuar la sustracción al sumar el negativo del sustraendo.

4 CÓDIGOS Y CONVERSIÓN DE CÓDIGO

La información que los humanos se transmiten se expresa por medio de un conjunto de símbolos. Cada uno de ellos constituye un *alfabeto*. Las letras A, B, ..., Z forman el alfabeto en el cual se expresa la lengua inglesa; los dígitos decimales conforman el alfabeto en el cual se expresa información numérica, etc. Para formar una palabra en inglés, se colocan letras específicas de extremo a extremo (*concatenadas* o *yuxtapuestas* en una *hiler*a).

La lengua inglesa puede considerarse como un *código*. A cada mensaje que se transmitirá (por ejemplo, *computadora*, *libro*, *paz*) se asigna una secuencia de letras (los dígitos en el alfabeto); el resultado es una *palabra de código*. El código corresponde a una colección que incluye todas estas palabras, a saber, la lengua inglesa.

Pensando de manera más general, un conjunto de símbolos constituye un alfabeto del cual estos símbolos son los dígitos. Las series de dígitos reciben el nombre de palabras. La asignación de una palabra de código a cada mensaje en un conjunto de mensajes constituye el código.

El sistema de números binarios es un alfabeto con sólo dos dígitos; es posible utilizarlo para codificar cualquier información deseada. El número de bits que se usará en cada palabra de código depende del número total de mensajes distintos que se van a transmitir en la información

deseada. Si el número de mensajes es exactamente igual a dos, entonces resulta suficiente una palabra de código de un solo bit.

Suponga que un objeto puede tener uno de cuatro colores únicos: rojo, amarillo, verde y azul. Es posible utilizar un código binario en el cual cada palabra tenga dos bits. Las cuatro combinaciones posibles de dos bits son 00, 01, 10 y 11. Cada palabra de código puede asignarse a uno de los colores, aunque no hay un criterio para asociar los colores con las palabras. De esta manera, el rojo puede asignarse a cualquiera de las palabras; si se efectúa una elección particular para el rojo, entonces al azul sólo se le asignará alguna de las tres palabras restantes; etc.

Ejercicio 9. Para el ejemplo del código de colores que acaba de indicarse, determine cuántos códigos diferentes existen.

Respuesta¹²

Considere ahora la perspectiva inversa. Suponga que cada palabra de código tiene n bits binarios. ¿Cuántos mensajes distintos pueden formarse? Este número corresponderá al número de series distintas que es posible formar con n bits, a saber 2^n . Para $n = 3$, por ejemplo, las $2^3 (= 8)$ palabras de 3 bits que pueden formarse son 000, 001, 010, 011, 100, 101, 110, 111.

En las siguientes secciones secundarias, estudiaremos diferentes códigos que se aplican en sistemas digitales en general, y en computadoras digitales en particular. En los últimos capítulos de este libro se estudiarán ciertos circuitos digitales que desempeñan un papel en los procesos de codificación y decodificación.

Decimal codificado en el sistema binario

Aunque casi todos los sistemas digitales operan con señales binarias, algunos sistemas efectúan aritmética en el sistema decimal. En consecuencia, existe la necesidad de expresar números decimales en algún código binario. Un posible código podría ser expresar cada número decimal mediante su equivalente binario; por ejemplo, el decimal 213 se codificaría como 11010101. Éste es un sistema muy inconveniente e ineficiente, que requiere tantas palabras de código como la cantidad de números decimales que es posible encontrar.

Un mejor esquema de codificación empieza con el reconocimiento de que cualquier número decimal está conformado por una serie de los 10 dígitos decimales. Si se asigna una palabra de código a cada dígito decimal, entonces cualquier número decimal puede codificarse como una serie de estas palabras. La codificación de los 10 dígitos decimales requiere un mínimo de 4 bits (confirme que 3 bits no es suficiente). Justo como en la codificación de colores que se mencionó antes, hay gran número de diferentes formas de asignar 4 bits a los dígitos decimales. Una elección particular consiste en codificar cada dígito decimal por medio de su equivalente binario de 4 bits. Este código particular recibe el nombre de sistema *decimal codificado en binario* o BCD. Puesto que el número de mensajes que puede expresarse con un código de 4 bits es $2^4 = 16$, y sólo hay 10 dígitos decimales, seis de las posibles palabras de código no se usan en el código BCD.

La figura 4 muestra varios códigos diferentes para los dígitos decimales que encuentran alguna aplicación; el código BCD aparece en la segunda columna.

Códigos ponderados

En el código BCD que acaba de presentarse, la posición de cada dígito binario corresponde a una potencia de 2. Afirmamos que los dígitos están *ponderados* por estas potencias de 2; los pesos son 8421, en ese orden. La suma de los pesos de todos los bits 1 en una palabra es igual al dígito

¹² $4! = 4 \times 3 \times 2 \times 1 = 24$.

Dígito decimal	BCD 8421	2421	84-2-1	Exeso de 3	Gray	Sólo activo (one hot)	Biquinario 5043210
0	0000	0000	00 0 0	0011	0010	100000000	0100001
1	0001	0001	01 1 1	0100	0110	010000000	0100010
2	0010	0010	01 1 0	0101	0111	001000000	0100100
3	0011	0011	01 0 1	0110	0101	000100000	0101000
4	0100	0100	01 0 0	0111	0100	000010000	0110000
5	0101	1011	10 1 1	1000	1100	000001000	1000001
6	0110	1100	10 1 0	1001	1101	000000100	1000010
7	0111	1101	10 0 1	1010	1111	000000010	1000100
8	1000	1110	10 0 0	1011	1110	000000001	1001000
9	1001	1111	11 1 1	1101	1010	000000000	1010000

Figura 4. Códigos binarios para los dígitos decimales.

decimal correspondiente (convéngase usted mismo de esto examinando los códigos de unos cuantos dígitos decimales).

En lugar de los pesos correspondientes a las potencias de 2, es posible asignar diferentes pesos a cada una de las posiciones de una palabra de 4 bits. Cada una de dichas asignaciones de peso generará un código ponderado diferente para los dígitos decimales; es posible incluso asignar pesos negativos. Unas cuantas posibilidades al lado del código ponderado 8421 (BCD) se muestran en las siguientes tres columnas de la figura 4.

Algo curioso se vuelve evidente al examinar los pesos en los diferentes códigos ponderados de la figura 4: la palabra de código para un dígito decimal no es necesariamente única. Por ejemplo, en el código 2421, 2 bits diferentes tienen el mismo peso. La palabra de código para el decimal 2 se da como 0010 en la columna correspondiente al código 2421 en la tabla 2. Sin embargo, la palabra 1000 representa también al decimal 2 en ese código.

Ejercicio 10. Construya un código ponderado para los dígitos decimales que tienen los pesos 642-3. Especifique todos los dígitos decimales que pueden representarse mediante dos palabras de código diferentes. *

Cuando hay una elección de palabras de código para representar un dígito decimal en un código ponderado particular, ¿cómo realizamos la elección? ¿Qué implicaciones tendrá la elección? En realidad, ¿cuál fue el criterio de elección que se utilizó en la figura 4? Examine las dos posibilidades para el decimal 2 en el código ponderado 2421: 0010 y 1000. Suponga que los bits 0 y 1 se intercambian en el primero de estos; el resultado es 1101, que corresponde al decimal 7 en el código 2421. Pero 7 es el complemento a nueve de 2. Examine ahora desde esta perspectiva el resto de las palabras de código para los dígitos decimales en el código 2421. Encontrará que el intercambio de ceros y unos para cada dígito decimal conduce al complemento a nueve de ese dígito. Un código de este tipo se denomina código de *autocomplementación*. Por lo general, un código de autocomplementación es aquél en el cual la palabra de código para el complemento a nueve de un dígito decimal d (a saber, $9 - d$) se obtiene sustituyendo cada bit binario por su complemento.

Ejercicio 11. Determine si el código BCD es de autocomplementación. Repita el proceso para el caso del código 2421 si se realizan las otras elecciones posibles de palabras de código binario para los decimales 2 y 7. Repita el proceso para el código 642-3 si se efectúan las otras elecciones posibles de palabras de código binario para los decimales 3 y 6.

Respuesta¹³

¹³ Para BCD, no. Para 2421, sí. Para 642-3, sí.

1 bit	2 bits	3 bits	4 bits
0	0 0	0 00	0 000
1	0 1	0 01	0 001
	1 1	0 11	0 011
	1 0	0 10	0 010
		1 10	0 110
		1 11	0 111
		1 01	0 101
		1 00	0 100
			1 100
			1 101
			1 111
			1 110
			1 010
			1 011
			1 001
			1 000

Figura 5. Generación del código Gray por medio de reflexión.

Código Gray

Son posibles muchos otros códigos para los dígitos decimales, de los cuales se listan unos cuantos en la figura 4. Algunos de éstos utilizan más de cuatro dígitos; tienen redundancias y resultan en consecuencia ineficientes para el cómputo. Sin embargo, podrían tener otras aplicaciones, como los dispositivos de salida de corrección o control de errores en un sistema digital. Algunos revisten principalmente un interés histórico. Por ejemplo, el código de exceso 3, como su nombre implica, se obtiene sumando 0011 a las palabras de código BCD para los dígitos decimales. Debido a que facilita el cálculo, el código de exceso 3 se utilizó comúnmente en las primeras computadoras. En la actualidad no se le emplea mucho para hacer cálculos.

Un código con una característica muy útil es aquél en el cual las palabras de código sucesivas difieren exactamente en una posición de bit.¹⁴ Se dice que este tipo de códigos son *cíclicos*. Un código cíclico que tiene sólo 2 bits, por ejemplo, es 00, 01, 11, 10. Sólo cambia un bit al ir de una palabra a la palabra adyacente, incluso cuando se regresa de la última palabra a la primera.

Un ejemplo específico de un código cíclico es un *código Gray*. La figura 5 ilustra el método para generar el código de $(n + 1)$ bits a partir del código de n bits. El código de 3 bits, por ejemplo, se obtiene *reflejando* el código de 2 bits alrededor de un eje dibujado debajo de la última palabra; luego, el bit más significativo de 0 se agrega arriba del eje y un msb de 1 debajo del eje. Debido a la manera en que se genera, tal código también recibe el nombre de código *reflejado*.

El código Gray completo de n bits tiene 2^n palabras. Si los dígitos decimales se van a codificar en un código Gray de 4 bits, ¿qué 10 de las 16 palabras debemos usar? Una de las muchas posibilidades es similar al código de exceso 3; éste omite las primeras tres y las últimas tres palabras de la última columna de la figura 5.

Código de siete segmentos

En muchas aplicaciones de sistemas digitales (relojes, radios, cronómetros, temporizadores, videos), la exhibición de los dígitos decimales desempeña una función importante. Ésta se consigue

¹⁴ Esto es en particular útil en los convertidores A/D (analógico-digital). Una función continua (analógica) se aproxima primero mediante escalones discretos. Esta información discreta se codifica posteriormente. La ambigüedad y los errores posibles se reducen si se asigna a los valores discretos adyacentes palabras de código adyacentes, esto es, palabras de código que difieran únicamente un bit.

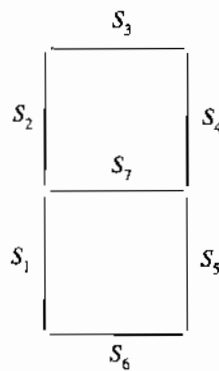


Figura 6. Indicador del código de siete segmentos.

con diodos emisores de luz (LED) o cristales líquidos (LC). Siete de estas fuentes luminosas se arreglan en la configuración estándar que se muestra en la figura 6; cada una puede activarse o desactivarse de manera independiente. Para exhibir uno de los dígitos decimales, debe activarse un subconjunto específico de los segmentos. Por ejemplo, para exhibir un 1, se iluminan los segmentos S_4 y S_5 ; los demás se desactivan.

Ejercicio 12. Suponga que los dígitos 1 y 0 representan los estados *activado* y *desactivado*, respectivamente. Elabore una tabla en la cual la primera columna liste los dígitos decimales. Los encabezados de las otras columnas (siete de ellas) serán los rotulados de segmento de S_1 al S_7 . Las entradas en cada renglón de la tabla serán los bits 0 y 1: 1 si el segmento correspondiente está *activado* y 0 si se encuentra *desactivado*. De ese modo, cada renglón en la tabla es una palabra de código de 7 bits que representa un dígito decimal. Para iluminar el dígito 1, por ejemplo, el renglón correspondiente en la tabla será 0 0 0 1 1 0 0. La tabla completa constituye el *código de siete segmentos*.

Códigos alfanuméricos

Todos los códigos presentados hasta ahora son representaciones de los dígitos decimales. Sin embargo, en muchas aplicaciones de procesamiento de datos resulta necesario manejar listas de miembros de una clase: clientes de una empresa de servicio público de electricidad, votantes organizados por distrito electoral, tenedores de pólizas de seguros, personas en la nómina, etcétera.

En esos casos, la representación de nombres, direcciones, designaciones de partido político, salarios y datos similares requiere un código para las letras del alfabeto, las marcas de puntuación, los dígitos decimales y otros símbolos. Un código binario que representa símbolos literales y numéricos, así como otros posibles símbolos especiales, recibe el nombre de *código alfanumérico*.

Ejercicio 13. a) Especifique el número mínimo de bits necesario en un código alfanumérico para codificar el alfabeto inglés (sólo mayúsculas) y los dígitos decimales. ¿Cuántos símbolos adicionales (por ejemplo, paréntesis, corchetes, marcas de puntuación) podría aceptar este código?
b) Haga lo mismo si se van a usar tanto letras mayúsculas como minúsculas. ♦

Un código de 7 bits puede representar $2^7 = 128$ símbolos. Esto resulta suficiente para los dígitos decimales, las letras minúsculas y mayúsculas del alfabeto inglés y otros 66 símbolos. Uno de los posibles códigos de 7 bits adoptado recibe el nombre de Código Estándar Americano para el Intercambio de Información (ASCII). (El ASCII de 8 bits para los símbolos estándares es simplemente el código ASCII de 7 bits precedido por un bms de 0. Es posible utilizar un bms de 1 para cualquier expansión futura de los símbolos estándares. Se adoptó un código de 8 bits debido a que

Carácter	Binario de 8 bits	Hexadecimal	Carácter	Binario de 8 bits	Hexadecimal
A	0100 0001	41	Espacio	0010 0000	20
B	0100 0010	42	\$	0010 0100	24
C	0100 0011	43	(0010 1000	28
D	0100 0100	44)	0010 1001	29
E	0100 0101	45	*	0010 1010	2A
F	0100 0110	46	+	0010 1011	2B
G	0100 0111	47	,	0010 1100	2C
H	0100 1000	48	-	0010 1101	2D
I	0100 1001	49	.	0010 1110	2E
J	0100 1010	4A	/	0010 1111	2F
K	0100 1011	4B	0	0011 0000	30
L	0100 1100	4C	1	0011 0001	31
M	0100 1101	4D	2	0011 0010	32
N	0100 1110	4E	3	0011 0011	33
O	0100 1111	4F	4	0011 0100	34
P	0101 0000	50	5	0011 0101	35
Q	0101 0001	51	6	0011 0110	36
R	0101 0010	52	7	0011 0111	37
S	0101 0011	53	8	0011 1000	38
T	0101 0100	54	9	0011 1001	39
U	0101 0101	55	=	0011 1101	3D
V	0101 0110	56			
W	0101 0111	57			
X	0101 1000	58			
Y	0101 1001	59			
Z	0101 1010	5A			

Figura 7. Código alfanumérico ASCII.

las computadoras por lo general manejan 8 bits en conjunto como una entidad llamada 1 *byte*.) La figura 7 muestra un subconjunto del código ASCII, tanto en forma binaria como en la forma hexadecimal más simple.

A continuación se presenta un ejemplo del uso del código ASCII en forma hexadecimal.

44 41 56 45 20 53 4D 49 54 48 2C 20 32 37 20 4D 41 49 4E 20 53 54 2E
 D A V E S M I T H , 2 7 M A I N S T .

5 DETECCIÓN Y CORRECCIÓN DE ERRORES

El proceso de generación de palabras de código para representar información se denomina *codificación* o *codificado*. Para que sea de utilidad, esta información codificada debe *procesarse* de ciertas maneras: requiere transmitirse a través de un canal de transmisión, almacenarse, recuperarse, etc. En cualquiera de estos procesos existe la posibilidad de que ocurran cambios en una señal debido a ruido, interferencia de otras señales, mal funcionamiento de los componentes del circuito o alguna otra causa.

En el caso de las señales digitales codificadas, el único error significativo se presenta cuando uno o más bits cambian de valor (0 a 1 o 1 a 0). La probabilidad de que ocurra un error simple de este tipo siempre es distinta de cero, aunque posiblemente pequeña. La probabilidad de que dos o más bits tengan simultáneamente valores erróneos es bastante menor. Hasta un primer orden, si es posible detectar todos los errores simples, podemos suponer que casi todos los errores posibles se han encontrado.

Códigos de detección de errores

Suponga que se va a transmitir algún mensaje. El primer paso consiste en codificarlo. Suponga que para este fin se usará el código BCD y que una de las palabras transmitidas es 0001. Suponga luego que ocurre un error y que uno de los bits en esta palabra cambia de valor. El mensaje recibido será entonces una de las siguientes cuatro posibilidades: 1001, 0101, 0011 o 0000. Cada una de éstas es una palabra de código BCD válida. Supusimos que recibiríamos el dígito decimal 1, pero recibimos erróneamente el dígito 9, 5, 3 o 0. Sin embargo, lo que es incluso más serio, no estamos conscientes de que haya habido un error. Cuando ocurre un error, sería muy útil que, como consecuencia, se tomara una acción —quizá activar una alarma o, mejor, provocar que el mensaje se repita de manera automática.

Suponga ahora que en lugar de un BCD se recurre a un código diferente, y que éste tiene la propiedad de que un error en cualquier bit convierte una palabra de código válida en una inválida, esto es, en un galimatías. Si se recibe una palabra de código inválida, sabemos en definitiva que ha ocurrido un error. Una palabra de este tipo se conoce como código de detección de errores (error simple).

Un código arbitrario de 4 bits para los dígitos decimales no es un código de detección de errores. Sin embargo, un código de este tipo puede convertirse en uno de detección de errores si se agrega un quinto bit, denominado bit de *paridad*. La paridad de una palabra de código se refiere al número de bits 1 en la palabra; la paridad (número de unos) es par o impar. Suponga que la paridad de cualquier palabra en un código determinado es par. Un error simple provocará que la paridad de la palabra se vuelva impar; si cualquier 0 se cambia por un 1; se incrementará el número de bits 1, o si un 1 se cambia a 0, se reducirá el número de bits 1. Observe, entre paréntesis, que *cualquier número impar* de errores originará el mismo resultado. (Un resultado similar se aplicaría si la paridad original de cada palabra en un código dado hubiera sido impar.)

Si se observa la paridad de las palabras en los datos recibidos en determinado caso y se encuentra que una palabra tiene paridad diferente de la que se supone, entonces debe haber ocurrido un error. En el código BCD la paridad de las palabras no es uniforme; la paridad es impar para ciertas palabras y par para otras. Sin embargo, un quinto bit (paridad) puede agregarse a cada palabra para conseguir que la paridad de todas las nuevas palabras de 5 bits sea la misma, ya sea par o impar. El caso correspondiente a la paridad impar, con el bit de paridad en la posición del lsb, se muestra en la última columna de la figura 8. Un código de estas características es un *código de detección de errores*. La columna a la mitad de la tabla produce otro código de detección de errores. Como su nombre lo indica, el código 2 de 5 genera todas las posibles formas de asignar dos bits 1 en una palabra de 5 bits; todas las palabras en este código tienen paridad par.

El que un código específico sea o no de detección de errores puede decidirse en términos del número de bits que deben cambiarse para transformar una palabra de código válida en una inválida. Definamos la *distancia* entre dos palabras de código como el número de bits en una palabra que debe complementarse para transformarla en una segunda palabra. Por ejemplo, la distancia entre 0011 y 0010 es 1; la distancia entre 0011 y 0100 es 3. En el código BCD la distancia entre dos palabras varía de 1 a 4. La *distancia mínima* de un código es el número más pequeño de bits mediante el cual varían dos palabras del código.

Ejercicio 14. Mediante el examen del código BCD y de los dos códigos de la figura 8, especifique la distancia mínima de cada uno.

Respuesta¹⁵

¹⁵ BCD, 1; BCD de paridad impar, 2; código 2 de 5, 2.

Dígito decimal	Código 2 de 5	BCD de paridad impar
0	00011	00001
1	00101	00010
2	00110	00100
3	01001	00111
4	01010	01000
5	01100	01011
6	10001	01101
7	10010	01110
8	10100	10000
9	11000	10011

Figura 8. Códigos de detección de errores.

Ejercicio 15. Demuestre que un código no puede ser de detección de errores si su distancia mínima es menor que 2. ♦

Suponga que la distancia mínima de cierto código es m . Suponga también que ocurren errores y que, en una palabra particular, se complementan hasta $m - 1$ bits. La palabra que se recibe será inválida y el error puede detectarse. Así, cuanto mayor sea la distancia mínima en un código, mayor será el número de errores simultáneos que es posible detectar.

Códigos de corrección de errores

En la sección anterior se demostró que mediante el diseño apropiado de un código con una distancia mínima de m , podríamos asegurar la posibilidad de detectar hasta $m - 1$ errores. ¡Esto no constituye a una hazaña! Una vez establecido que ha ocurrido un error, sería todavía mejor corregirlo y reconstruir la palabra corregida. Si un código tiene la propiedad de que la palabra de código correcta siempre puede deducirse a partir de la palabra errónea, afirmamos que se trata de un *código de corrección de errores*.

La figura 9 muestra un código de corrección de errores simple. Verifique que la distancia mínima en este código es 3. Si ocurren dos errores simultáneos, la palabra que resulta sigue a una distancia de 1 para ser una palabra de código válida. Esto quiere decir que la palabra de código errónea no puede interpretarse como una palabra de código válida, por lo que es posible detectar un error doble. Ahora supongamos que ocurre un error simple en una de las palabras de código; digamos que la palabra transmitida correcta 00011 (mensaje M_4) se recibe como cualquiera de las siguientes palabras erróneas: 10011, 01011, 00111, 00001 o 00010. Comparándolas con otras palabras en el código se observa que cada palabra recibida está a una distancia de al menos 2 de cualquiera de ellas. (¡Hágalo!) En consecuencia, si se recibe una de estas palabras erróneas y se supone que sólo ha ocurrido un error simple, éste no pudo haber provenido de otra palabra más que de 00011 (M_4). La última palabra correcta se reconstruye complementando el único bit que sabemos que es erróneo. (El bit apropiado se detecta comparando la palabra de código errónea con la palabra de código válida.)

Mensaje	Código
M_1	11111
M_2	00100
M_3	11000
M_4	00011

Figura 9. Código de corrección de errores.

De acuerdo con la explicación anterior, resulta claro que el uso de un código de distancia mínima 3 permite *corregir* cualquier error simple o *detectar* cualquier error doble. El mismo concepto puede aplicarse evidentemente a códigos con distancias mínimas mayores.

Códigos Hamming¹⁶

La sección anterior sólo introdujo el concepto de códigos de corrección de errores. Ahora necesitamos especificar un código de este tipo que resulte útil y práctico. Existen muchas posibilidades; la clase de códigos que se describirán aquí reciben el nombre de *códigos Hamming*¹⁷. Para empezar, suponga que los mensajes se codifican en BCD o en otro código de 4 bits. Se van a agregar varios bits de *verificación de paridad* a las palabras de código, dispersos entre los bits del *mensaje*. La posición de cada bit en la palabra resultante se designa contando hacia la derecha a partir del bit más significativo, cuya posición es 1.

¿Cuántos bits de verificación de paridad deben usarse, y cómo deben distribuirse sus posiciones en la palabra resultante? La primera pregunta se contesta observando que deben existir suficientes bits de verificación de paridad para lograr que la distancia mínima sea al menos 3 a fin de que el código pueda ser de corrección de errores. En la sección precedente se observó que agregar un solo bit de paridad a un código BCD genera un código de distancia mínima 2. Convertir un código de 4 bits en uno de corrección de errores requiere un mínimo de 3 bits adicionales. Si se utilizan códigos de n bits (con $n > 4$), entonces podrían necesitarse más de 3 bits de verificación de paridad.

Ejercicio 16. Determine el número de posibles palabras de código de seis dígitos necesarias para que los errores simples sean únicos, para demostrar que al menos se necesitan 3 bits de verificación de paridad para convertir un código de 4 bits en uno de corrección de errores.

Respuesta¹⁸

Volvamos a la segunda pregunta: ¿cuántos bits de verificación de paridad deben distribuirse en la palabra? Una palabra de código de 7 bits que consta de 4 bits de mensaje y 3 bits de verificación de paridad puede escribirse del modo siguiente:

$$d_1 d_2 d_3 d_4 d_5 d_6 d_7$$

donde d representa a los dígitos. La idea es realizar tres verificaciones de paridad eligiendo de tres maneras diferentes 4 de los 7 bits. El resultado de estas tres verificaciones será una palabra de 3 bits $c_1 c_2 c_3$. Cada bit c_i será 1 si se detecta un error mediante la verificación correspondiente y 0 si no. La palabra de 3 bits se denomina *síndrome*; su equivalente decimal será la posición del dígito en la palabra de código de 7 bits en el cual ha ocurrido el error. Desde luego, el síndrome 000 significa que no ha ocurrido error.

Síndrome: $c_1 c_2 c_3$

¹⁶ El material en esta sección en ocasiones se describe como "avanzado". En realidad no está más allá de su capacidad, aunque puede omitirlo si lo desea.

¹⁷ Éstos reciben ese nombre en honor a Richard W. Hamming (1915-1997), colaborador durante 30 años de los Laboratorios Bell, que los propuso por primera vez en 1947. Después de retirarse de Bell, emprendió otra carrera de 21 años en la Naval Postgraduate School. Murió el último día de 1997.

¹⁸ Con sólo 2 bits adicionales, cada palabra de código tendría 6 bits. Los 10 dígitos decimales requerirían 10 palabras (correctas). Los errores simples ocurrirían en cada bit de una palabra; por ello cada palabra de código válida tendría 6 distintas palabras de código inválidas, lo que implicaría un total requerido de $10 + 60 = 70$ palabras de código diferentes de seis dígitos. Sin embargo, para seis dígitos sólo hay $2^6 = 64$ combinaciones. *Conclusión:* El uso de 2 bits de verificación de paridad únicamente resulta insuficiente para formar un código de corrección de errores a partir de un código de 4 bits de los dígitos decimales; al menos se necesitan tres bits de verificación de paridad.

Bit de verificación de paridad	Paridad par en posiciones	Dígitos del mensaje	Paridad	Valor de p_i
p_1	1, 3, 5, 7	m_1, m_2, m_4	Par	$p_1 = 0$
p_2	2, 3, 6, 7	m_1, m_3, m_4	Impar	$p_2 = 1$
p_3	4, 5, 6, 7	m_2, m_3, m_4	Par	$p_3 = 0$

a)

b)

Figura 10. Determinación de bits de paridad en el ejemplo.

Las posiciones de los 3 bits de verificación de paridad en la palabra de 7 bits se elegirán de manera que el síndrome correspondiente tenga un único bit distinto de cero: 001, 010 o 100 (decimal 1, 2 o 4). Cada uno de estos bits será, en consecuencia, independiente de los otros. Con los bits de verificación de paridad en las posiciones 1, 2 y 4 en la palabra de 7 bits precedente, la distribución de los bits de mensaje y de verificación de paridad es como sigue:

$$\begin{array}{l} \text{Posiciones: } 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \\ \text{Bits: } p_1 \quad p_2 \quad m_1 \quad p_3 \quad m_2 \quad m_3 \quad m_4 \end{array} \quad (6)$$

Suponga que c_3 es 1; esto significa que una verificación de paridad indica un error. Entonces, sin que importen los valores de c_1 y c_2 , el síndrome debe ser impar (1, 3, 5 o 7). Los dígitos en las posiciones 3, 5 y 7 son dígitos de mensaje. Suponga que p_1 se elige en cada palabra de código de manera que los dígitos en las posiciones 1, 3, 5 y 7 (p_1, m_1, m_2, m_4) tengan paridad par. Si ocurre un error simple en cualquiera de estas posiciones, la verificación de paridad producirá un síndrome en el cual c_3 es 1.

De modo similar, considere que c_2 es 1; si c_1 y c_3 toman sus cuatro posibles combinaciones, los valores posibles del síndrome corresponden a 2, 3, 6 y 7 (confirme estos valores). La posición 2 la ocupa el bit de verificación de paridad p_2 mientras que las otras tres posiciones corresponden a los dígitos del mensaje. Como en el caso anterior, elegimos p_2 en cada palabra de código de manera que p_2, m_1, m_3 y m_4 tengan en conjunto paridad par. Si ocurre un error en cualquiera de estos bits, entonces la verificación de paridad producirá $c_2 = 1$.

En el caso final, $c_1 = 1$ puede ocurrir para los síndromes 4, 5, 6 y 7. Por consiguiente, se elige p_3 en cada palabra de código de manera que p_3, m_2, m_3 y m_4 tengan en conjunto paridad par. Esta vez un error en cualquiera de estos bits generará un síndrome con $c_1 = 1$.

El análisis precedente se resume en las tablas de la figura 10; los bits de verificación de paridad se eligen de manera que haya paridad par entre los conjuntos de 4 bits que se muestran en la columna derecha de la figura 10a.

El código de Hamming para el BCD construido de esta manera aparece en la figura 11. Comparando las palabras de código, es fácil ver que su distancia mínima es 3.

Para ilustrar la generación de los bits de verificación de paridad, considere la palabra de código correspondiente a 5: $m_1 m_2 m_3 m_4 = 0101$. Las paridades de los dígitos del mensaje y, en consecuencia, las elecciones de las p_i , se muestran en la figura 12. La palabra de código que resulta es 0100101; esto confirma la palabra de código para el dígito decimal 5 en la figura 11.

La característica de corrección de errores del código de Hamming opera del siguiente modo. Suponga que la palabra transmitida es 1110000 (decimal 8), pero que la palabra recibida es 1110010, con un solo bit en error. Las tres verificaciones de paridad producen el resultado en la figura 12. Así, se detecta un error en la posición 6. Éste se corrige complementando el dígito en la posición 6 (m_3).

$$\begin{array}{ll} \text{Palabra transmitida} & 1110000 \\ \text{Palabra recibida} & 1110010 \\ \text{Síndrome} & 110 \end{array} \quad (7)$$

Dígito decimal	Posición: Dígito:	1 p_1	2 p_2	3 m_1	4 p_3	5 m_2	6 m_3	7 m_4
0		0	0	0	0	0	0	0
1		1	1	0	1	0	0	1
2		0	1	0	1	0	1	0
3		1	0	0	0	0	1	1
4		1	0	0	1	1	0	0
5		0	1	0	0	1	0	1
6		1	1	0	0	1	1	0
7		0	0	0	1	1	1	1
8		1	1	1	0	0	0	0
9		0	0	1	1	0	0	1

Figura 11. Código de Hamming para los dígitos decimales basados en BCD.

Verificador de paridad en las posiciones:	Dígitos recibidos	Paridad	c_i
4, 5, 6, 7	0010	Impar	$c_1 = 1$
2, 3, 6, 7	1110	Impar	$c_2 = 1$
1, 3, 5, 7	1100	Par	$c_3 = 0$

Figura 12. Síndrome del código de Hamming de ejemplo.

El tema de los códigos de detección y de corrección de errores es fascinante. Sin embargo, no proseguiremos con esta materia en otra parte del libro.

RESUMEN Y REPASO DEL CAPÍTULO¹⁹

En este capítulo se definieron los conceptos básicos y los fundamentos matemáticos en los que se basa el diseño lógico. Lo que sigue es un resumen del capítulo.

- Sistema numérico binario
- Sistemas numéricos octal y hexadecimal
- Conversión de un sistema numérico a otro
- Conversión al sistema decimal a partir del sistema binario
- Conversión del sistema decimal al binario
- Conversión del sistema octal (o hexadecimal) al binario
- Representación de números negativos
- Bases de la aritmética binaria
 - Suma
 - Resta
 - Multiplicación
 - División

¹⁹ Al final de cada capítulo se ofrece un resumen y un repaso del mismo. En cada caso se revisan todos los conceptos y principios importantes, así como los procedimientos y los dispositivos del capítulo. Al leer cada entrada en el resumen del capítulo, usted mismo debe explicar, como lo haría ante un alumno, las ideas, definiciones, principios, teoremas, dispositivos, algoritmos, diagramas y otras características importantes. Cuando no se encuentre seguro de algo, regrese a las páginas correspondientes y repase el contenido hasta asimilarlo perfectamente.

- Códigos numéricos
- Códigos ponderados
- Números decimales codificados en binario
- Código Gray
- Código de siete segmentos
- Código alfanumérico ASCII
- Detección de errores y códigos de detección de errores
- Códigos de corrección de errores
- Códigos de Hamming
- Verificación de paridad
- Un síndrome

PROBLEMAS

1. Considere los siguientes pares de números decimales.

(18, 7) (75, 28) (11.3, 23.5) (31.25, 17.58)

- a. Convierta cada par de decimales al sistema binario.
 - b. Encuentre la suma de cada par de números binarios; compruebe el resultado determinando la suma decimal y convirtiéndola al sistema binario.
 - c. Encuentre la diferencia de cada par binario; compruebe el resultado convirtiendo la diferencia decimal al sistema binario.
 - d. Determine el producto de cada par binario; compruebe el resultado convirtiendo el producto decimal al sistema binario.
2. Exprese los siguientes números irracionales en el sistema binario con suficientes dígitos en la parte fraccionaria de manera que la exactitud no sea menor de 4 lugares decimales en el equivalente decimal: a) π ; b) base de los logaritmos naturales e ; c) $e^{-\pi/2}$; d) $\sqrt{15}$.
 3. Sean los siguientes números decimales: 347.8, 2989, 625.7.

- a. Convierta cada número al sistema binario.
- b. Convierta cada número del decimal al sistema octal. Compruebe el resultado convirtiendo la forma binaria a la octal.
- c. Convierta cada número del sistema decimal al hexadecimal. Compruebe el resultado convirtiendo la forma binaria a la hexadecimal.

4. Sea el siguiente conjunto de números binarios:

10111, 110100.1, 111001.01, 1100101.101

- a. Convierta cada uno en un número decimal.
- b. Convierta cada uno en un número octal.
- c. Convierta cada uno en un número hexadecimal.

5. Sea el siguiente conjunto de números octales:

247, 153.4 374.25, 5120.7

- a. Convierta cada uno en un número binario.
 - b. Convierta directamente cada uno en un número decimal; compruebe el resultado convirtiendo el equivalente binario en decimal.
 - c. Convierta directamente cada uno en hexadecimal; verifique el resultado convirtiendo el equivalente binario en hexadecimal.
6. La base en la cual se representan los números en las siguientes operaciones (al lado izquierdo de la igualdad) es b . Los números en el miembro derecho de cada igualdad son decimales. Encuentre todos los valores posibles de la base b limitados a los sistemas numéricos descritos en este capítulo.

- a. $\sqrt{121} = 8$
 b. $14.2 \times 30 = 510$
 c. $346 + 543 = 1111$
7. Las conversiones entre sistemas numéricos descritas en la sección 3 se formulan convenientemente con el fin de utilizar la aritmética decimal. Describa cómo convertir directamente de cualquier base b_1 a otra base b_2 sin usar números decimales. (El método no debe incluir la conversión de la base b_1 en decimal.)
8. Efectúe las siguientes operaciones aritméticas en sistema binario.
- a. $1001 + 0110/0011$
 b. $(0110 - 0101) \times 1100$
 c. $(0010 + 0001)^2$
9. Efectúe las siguientes operaciones de multiplicación binarias. Utilice tantos bits como sea necesario para representar el resultado.
- a. 100010×001010
 b. 001100×011001
 c. 000100×010101
10. Efectúe las siguientes operaciones de división binarias. Represente cada resultado como un cociente y el residuo.
- a. $001100/0101$
 b. $010010/0010$
 c. $110011/0100$
11. a. Escriba los complementos a uno de los siguientes números binarios.
- | | | |
|---------|---------|--------|
| 0011001 | 1110011 | 111111 |
| 1001011 | 1010101 | 000001 |
- b. Escriba los complementos a dos de los mismos números.
12. Expresé los siguientes números decimales en la representación numérica de complemento a dos.
- 36, 101, -49, -75
13. Especifique el rango de números decimales que puede encontrarse en la representación numérica de complemento a dos utilizando una longitud de palabra de 8 bits.
14. Utilizando la representación numérica de complemento a dos limitada a una longitud de palabra de 8 bits, efectúe las siguientes operaciones aritméticas. (Los números dados son decimales.) En cada caso, utilice únicamente el resultado de la operación aritmética (no su conocimiento del equivalente decimal), especifique si el entero resultante es positivo o negativo y describa cómo lo determinó.
- a. $76 + 12$ b. $83 + 60$ c. $75 - 203$
 d. $-28 - 64$ e. $37 + 80$ f. $253 - 182$
15. Encuentre el equivalente decimal de los siguientes números de complemento a dos.
- a. 11111111 b. 10000001 c. 01010101
 d. 10011100 e. 01110000 f. 10101010
16. ¿Bajo qué condición puede ocurrir desbordamiento si un número positivo de complemento a dos se agrega a un número negativo de complemento a dos?
17. Reste el número binario 0110 de sí mismo sumándole su complemento a uno. ¿Es el resultado que esperaba?
18. Supongamos que $V_{2c}(a_n a_{n-1} a_{n-2} \dots a_1 a_0)$ representa el valor de la secuencia binaria $a_{n-1} a_{n-2} \dots a_1 a_0$ en la representación numérica de complemento a dos.

- a. Confirme la siguiente propiedad: Si el bit de signo a_{n-1} se repite cualquier número de veces, el valor no cambia. Esto es,

$$V_{2c}(a_{n-1}a_{n-1}\dots a_{n-1}a_{n-2}a_{n-3}\dots a_1a_0) = V_{2c}(a_{n-1}a_{n-2}\dots a_1a_0)$$

Esto se conoce como propiedad de *extensión del signo*.

- b. Las siguientes secuencias están en representación de complemento a dos. Utilizando la propiedad de la extensión del signo, muestre cómo sumaría cada par de secuencias; verifique sus respuestas.

101 y 0101001, 0101 y 0101001, 101 y 1101001

- c. Demuestre la siguiente propiedad:

$$V_{2c}(a_{n-1}a_{n-2}\dots a_1a_0) = 2^0(0 - a_0) + 2^1(a_0 - a_1) + 2^2(a_1 - a_2) + \dots + 2^{n-1}(a_{n-2} - a_{n-1})$$

19. El *algoritmo de Booth* acepta dos números en la representación de complemento a dos y calcula su producto, también en complemento a dos. Proceda de la manera siguiente:

- a. Agregue un 0 al multiplicador en su extremo menos significativo. Después descomponga el resultado en dos grupos traslapantes de 2 bits. De esta forma, si el multiplicador es $a_{n-1}a_{n-2}\dots a_1a_0$ en orden ascendente, los grupos son

$$(a_0, 0), (a_1, a_0), (a_2, a_1), \dots, (a_{n-1}, a_{n-2}),$$

- b. Asigne el valor inicial de cero al *producto parcial* (pp).
 c. Si un grupo de 2 bits es (0,1), entonces incremente pp mediante el multiplicando; si un grupo de 2 bits es (1,0) entonces reduzca pp mediante el multiplicando; si un grupo de 2 bits es (0,0) o (1,1), entonces mantenga invariable pp.
 d. Duplique el multiplicando.
 e. Repita los pasos c y d hasta que todos los grupos de 2 bits, en orden ascendente, se agoten.

Con base en el resultado del problema 18c, demuestre que el resultado del algoritmo de Booth representa el producto del multiplicador y el multiplicando originales.

20. Demuestre el criterio de detección de desbordamiento para la adición en la representación numérica de complemento a dos —esto es, el resultado está fuera de rango si y sólo si el acarreo de entrada del bit de signo es diferente del acarreo de salida del bit de signo.
21. Sume los siguientes números binarios, suponiendo que son
- Números de complemento a uno.
 - Números de complemento a dos.

Identifique casos donde ocurra un desbordamiento aritmético.

$$\begin{array}{r} 0011 \\ +0011 \\ \hline \end{array} \quad \begin{array}{r} 1001 \\ +0101 \\ \hline \end{array} \quad \begin{array}{r} 1000 \\ +1010 \\ \hline \end{array} \quad \begin{array}{r} 0101 \\ +1100 \\ \hline \end{array} \quad \begin{array}{r} 0111 \\ +0010 \\ \hline \end{array}$$

22. ¿Es asociativa la suma de números binarios de precisión fija con signo? En otras palabras, ¿para números de complemento a dos de 8 bits, $(A + B) + C = A + (B + C)$? (Sugerencia: ¿qué sucede si hay desbordamiento?)
23. Represente cada número decimal a la izquierda en cada uno de los códigos de la derecha.

Número	Código
43	Binario
257	BCD
5823	643-2
	Exeso de 3
	Gray

24. Encuentre un código de autocomplementación para los dígitos decimales utilizando cada uno de los siguientes pesos.
- a. 443-2 b. 3321 c. 731-2 d. 87-4-2
25. Cuatro mensajes se codifican en las siguientes palabras de código.
- M_1 : 01101, M_2 : 10011, M_3 : 00110, M_4 : 11000
- a. Determine la distancia mínima de este código.
- b. Las siguientes palabras se reciben y se sabe que únicamente ha ocurrido un error de un bit. Especifique el mensaje correcto en cada caso.
- | | | |
|-------|-------|-------|
| 00010 | 11101 | 01001 |
| 11010 | 00011 | 10010 |
| 00111 | 11010 | 10110 |
26. Se transmiten palabras codificadas en el código de Hamming. Se reciben las siguientes palabras.
- | | | | |
|---------|---------|---------|---------|
| 0101000 | 0011101 | 1100100 | 1100110 |
| 1110011 | 1111001 | 1101001 | 1000010 |
- a. Si cualquiera de las palabras es correcta, especifique el dígito decimal correspondiente.
- b. Si cualquiera de las palabras tiene un error simple, especifique el bit erróneo y especifique el dígito decimal correcto.
- c. Especifique cualquiera de las palabras recibidas que tiene un error doble.
- d. Describa cómo detectar un error triple.
27. a. Suponga que un código tiene una distancia mínima de n , donde n es impar. Si se agrega un bit de paridad adicional a cada palabra de código para establecer paridad par sobre la palabra de código, describa cómo afectará esto la distancia mínima del código.
- b. Repita al inciso a), considerando que n es par.
- c. Repita el inciso a), considerando que el bit de paridad añadido produce paridad impar.
- d. Repita el inciso a), suponiendo que n es par y que el bit de paridad añadido establece paridad impar.
28. a. De acuerdo con los resultados del problema 26, sugiera y justifique un método para construir un código de corrección de error simple y de detección de error doble (SEC-DED) a partir de un código de Hamming de detección de error simple.
- b. Ejemplifique el método construyendo un código SEC-DED, en el cual cada palabra de código contiene un mensaje de 3 bits.
29. Encuentre si es cierta o falsa la afirmación de que, para un código de detección de errores m , debe ser posible especificar un algoritmo de decodificación tal que si la palabra recibida contiene exactamente j errores, para cada j desde 1 hasta m , el algoritmo indica que la palabra recibida tiene un error.
30. Encuentre si es cierta o falsa la afirmación de que es posible generar, para cualquier código de detección de errores m , un algoritmo de decodificación que señale que una palabra recibida contiene exactamente j errores cada vez que se lleve a cabo el algoritmo, para cada j entre 1 hasta m , aunque no sea capaz de identificar las posiciones de error.
31. Encuentre si es cierta o falsa la afirmación de que si han ocurrido m o menos errores en un código de corrección de errores m , es posible generar un algoritmo de decodificación que especifique cuántos errores han ocurrido y en qué posiciones.
32. Encuentre si es cierta o falsa la afirmación de que, para un código de corrección de errores m , si un algoritmo de decodificación especifica que hay exactamente j errores en una palabra recibida, con j de 1 a m , entonces realmente han ocurrido j errores.
33. Dada una palabra de código con una distancia mínima de 4, describa las circunstancias en las que usted desearía usar el código para detección triple de errores y cuándo para la corrección de error simple y la detección de errores dobles (SEC-DED).

34. Encuentre si son ciertas o falsas las siguientes afirmaciones.
- Para un código de distancia mínima d , no es posible detectar *algún* error de d bits.
 - Para un código de distancia mínima d , no es posible detectar *cualquier* error de d bits.
35. En el caso de un código de Hamming de corrección de un error simple, suponga que el número de bits de mensaje, m , y el número de bits de paridad, k , son tales que $2^k \geq m + k + 1$. Especifique si la indicación de la posición del error obtenida a partir de los verificadores de paridad en la palabra recibida se refiere a una posición no existente, esto es, una posición mayor que $m + k$. Explique por qué una indicación de este tipo es imposible o el significado físico de dicha indicación.
36. Suponga que se van a elegir los bits de verificación de paridad en un código de Hamming para BCD de tal forma que la paridad de las tres palabras que se van a verificar sea impar en lugar de par. Construya una tabla de código Hamming similar a la figura 10b del texto para detectar y corregir un solo error en el código transmitido.
37.
 - Suponga que la distancia mínima de un código es un número impar n . Se va a agregar un bit de paridad adicional a cada palabra de código para establecer paridad par en la palabra de código completa. ¿Qué efecto tendrá esto en la distancia mínima del código? Justifique su respuesta.
 - Repita el inciso a), pero considerando que n es par.
 - Repita el inciso a), considerando que el bit de paridad adicional establece paridad impar.
 - Repita el inciso a), tomando en cuenta que n es par y que el bit de paridad agregado establece paridad impar.
38.
 - Con base en el resultado del problema 37, sugiera un método para construir un código de corrección de error simple y de detección de errores dobles (SEC-DED) a partir de un código de Hamming de corrección de un error simple. Justifique su método.
 - Ilustre su método construyendo un código SEC-DED en el cual cada palabra de código contenga un mensaje de 3 bits.
 - Se va a diseñar un decodificador para el código SEC-DED construido en el inciso b) para usarse en el extremo de recepción. El decodificador recibe la palabra entrante y emite una de estas tres salidas:
 - MSG: Contiene el mensaje correcto de 3 bits cuando no ha ocurrido ningún error o error corregible;
 - CE: Indica un error corregible y que MSG contiene el mensaje corregido;
 - UE: Indica un error incorregible; MSG se va a descartar en este caso.
 - Analice lo que ocurre en el decodificador del inciso c) si la parte recibida contiene un error triple.
 - Suponga que el código construido en el inciso b) se va a usar para la detección de error triple en vez de la corrección de un error simple y de la detección de error doble. ¿Cuál sería un conjunto apropiado de salidas para el decodificador del extremo receptor?
39. Encuentre si son ciertas o falsas las siguientes afirmaciones.
- En un código de detección de m errores, debe ser posible proponer un algoritmo de decodificación tal que, para toda j desde 1 hasta m , si la palabra recibida contiene j errores, el algoritmo indica que la palabra recibida tiene errores.
 - En el caso de un código de corrección de m errores, debe ser posible proponer un algoritmo de decodificación tal que, para toda j desde 1 hasta m , si la palabra recibida contiene exactamente j errores, el algoritmo señala que la palabra recibida tiene exactamente j errores —aunque quizá no sea posible identificar la posición de los errores.
 - En un código de corrección de m errores, existe un algoritmo de decodificación tal que, si ha ocurrido un número de errores de hasta m , el algoritmo indica exactamente cuántos y en qué posiciones.
40. Suponga que en un código de Hamming de corrección de un error, el número de bits de mensaje m y el número de bits de paridad k son tales que $2^k > m + k + 1$. Establezca si la indicación de la posición del error, obtenida a partir de los verificadores de paridad en la palabra recibida, se refiere a una posición no existente (es decir, mayor que $m + k$). Explique por qué es imposible una indicación de este tipo o el significado físico de tal indicación.

41. a. Suponga que la distancia mínima de un código es un número impar n . Un bit de paridad adicional se va añadir a cada palabra de código para establecer paridad par en la palabra de código completa. ¿Qué efecto tendrá esto en la distancia mínima del código? Justifique su respuesta.
- b. Repita el inciso a), considerando que n es par.
- c. Repita el inciso a), considerando que el bit de paridad agregado establece paridad impar.
- d. Repita el inciso a), considerando que n es par y que el bit de paridad agregado establece paridad impar.

Álgebra de conmutadores y compuertas lógicas

La palabra *álgebra* en el título de este capítulo anuncia el uso de más matemáticas. Sin duda, algunos de los lectores tienen la inquietud de abordar ya el diseño digital en vez de adquirir mayor conocimiento matemático. Sin embargo, por su experiencia en ingeniería y ciencias, saben que las matemáticas constituyen un requerimiento básico en todos los campos de estas áreas. Del mismo modo que pensar requiere del conocimiento de un lenguaje en el cual sea posible formular los conceptos, cualquier campo de ingeniería y de ciencia necesita conocimiento de ciertos temas matemáticos en términos de los cuales los conceptos relativos puedan expresarse y comprenderse.

La base matemática para los sistemas digitales es el *álgebra booleana*.¹ Este capítulo inicia con una breve exposición del álgebra que establece el cimiento para presentar los bloques constitutivos de los circuitos digitales más adelante en el capítulo.

1 ÁLGEBRA BOOLEANA

El álgebra booleana, al igual que cualquier otra estructura matemática o sistema algebraico axiomático, puede caracterizarse especificando varias cuestiones fundamentales:

1. El *dominio* del álgebra, esto es, el conjunto de *elementos* sobre los cuales se define el álgebra.
2. Un conjunto de *operaciones* que se van a efectuar sobre los elementos.
3. Un conjunto de *postulados*, o *axiomas*, aceptados como premisa sin demostración.
4. Un conjunto de consecuencias denominado *teoremas*, *leyes* o *reglas*, los cuales se deducen de los postulados.

Como en cualquier área de las matemáticas, es posible iniciar de diferentes conjuntos de postulados y de igual modo llegar a la misma estructura matemática. Lo que se prueba como un teorema a partir de un conjunto de postulados puede considerarse como un postulado en otro conjunto, y lo que se postuló en el primer conjunto se demuestra como un teorema a partir de otro conjunto de postulados. Así, ¿cómo elegimos los postulados? Resulta claro que uno de los requerimientos para un conjunto de postulados es la *consistencia*. Ésta no haría que las consecuencias de un postulado contradijeran las de otro. Otro requerimiento que se establece a menudo es la *independencia*. Sin embargo, la independencia implica el objetivo usual de terminar con un conjunto *mínimo* de postulados que sigue permitiendo la deducción de todos los teoremas. De ese modo, mientras una regla matemática sea consistente con las otras, es posible agregarla como

¹ Esta designación proviene de su creador, el británico George Boole, quien publicó un trabajo titulado *An Investigation of the Laws of Thought* en 1854. Este tratado fue una exposición fundamental y sistemática de la lógica. El libro permaneció en la oscuridad durante muchas décadas.

Tabla 1 Postulados de Huntington

-
1. **Cierre.** Existe un dominio B que tiene al menos dos elementos distintos y dos operadores binarios $(+)$ y (\cdot) tales que:
 - a. Si x e y son elementos, entonces $x + y$ es un elemento.
La operación efectuada por $(+)$ recibe el nombre de *suma lógica*.
 - b. Si x e y son elementos, entonces $x \cdot y$ es un elemento.
La operación efectuada por (\cdot) recibe el nombre de *multiplicación lógica*.
 2. **Elementos identidad.** Sea x un elemento en el dominio B .
 - a. Existe un elemento 0 en B , llamado el *elemento identidad* con respecto a $(+)$, que tiene la propiedad $x + 0 = x$.
 - b. Existe un elemento 1 en B , denominado el *elemento identidad* con respecto a (\cdot) , que tiene la propiedad de que $x \cdot 1 = x$.
 3. **Ley conmutativa**
 - a. Ley conmutativa con respecto a la suma: $x + y = y + x$.
 - b. Ley conmutativa con respecto a la multiplicación $x \cdot y = y \cdot x$.
 4. **Ley distributiva**
 - a. La multiplicación es distributiva sobre la suma: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
 - b. La suma es distributiva sobre la multiplicación: $x + (y \cdot z) = (x + y) \cdot (x + z)$
 5. **Complementación.** Si x es un elemento en el dominio B , entonces existe otro elemento x' , el *complemento* de x , que satisface las propiedades:
 - a. $x + x' = 1$
 - b. $x \cdot x' = 0$
 El complemento x' efectúa la operación de *complementación* sobre x .
-

un postulado sin peligro. Sin embargo, si depende de los postulados previos, la regla agregada resulta derivable de ellos y no es necesario considerarla como un postulado.

Los postulados que adoptaremos aquí se conocen como *postulados de Huntington* y se presentan en la tabla 1.² Estúdielos cuidadosamente. Advierta que el álgebra booleana es similar al álgebra ordinaria en algunos aspectos, así como diferente en otros. Por ejemplo, la ley distributiva de la adición (postulado 4b) no es válida para el álgebra ordinaria, ni lo es la operación complemento (postulado 5). Por otro lado, las operaciones de resta y división del álgebra ordinaria no existen en el álgebra booleana.

El conjunto de elementos en el álgebra booleana se denomina su *dominio* y se designa con la letra B . Una *operación m -aria* en B es una regla que asigna a cada conjunto ordenado de m elementos un elemento único de B . De esta forma, una operación *binaria* implica un *par* ordenado de elementos, y una operación *unaria* implica sólo un elemento. En el álgebra booleana se definen dos operaciones binarias (suma lógica y multiplicación lógica) y una operación unaria (complementación). Pueden existir muchas álgebras booleanas con diferentes conjuntos de elementos. El término *álgebra booleana* es una forma genérica de referirse a todas ellas.

Principio de dualidad

Un examen de los postulados de Huntington revela cierta simetría: los postulados vienen en pares. Uno de los postulados en cada par puede obtenerse de otro:

- Intercambiando los dos operadores binarios, e
- Intercambiando los dos elementos identidad cuando aparecen en forma explícita.

De tal manera, es posible obtener una de las leyes conmutativas de la otra intercambiando los operadores $(+)$ y (\cdot) . Lo mismo es cierto en el caso de las dos leyes distributivas. En consecuencia, cualesquiera resultados que sea posible deducir de los postulados deben permanecer válidos si

² Los formuló el matemático inglés E. V. Huntington, quien intentó sistematizar el trabajo de George Boole exactamente 50 años después de la publicación del tratado de este último.

- los operadores (+) y (•) se intercambian, y
- los elementos identidad 0 y 1 se intercambian.

Esta propiedad del álgebra booleana se conoce como el *principio de dualidad*. Siempre que algún resultado (teorema) se deduce de los postulados, se recurre al principio de dualidad como prueba del teorema dual.

Teoremas fundamentales

Estableceremos ahora varias consecuencias (teoremas, reglas o leyes) que siguen de los postulados de Huntington y del principio de dualidad. Las demostraciones se efectuarán paso a paso, con una justificación explícita para cada uno de ellos al hacer referencia al postulado correspondiente o al teorema demostrado con anterioridad. Dos de los métodos generales de prueba que se utilizan en matemáticas son:

- Demostración por contradicción.
- Demostración por el principio de inducción (matemática).

Una prueba de contradicción se realiza suponiendo que el opuesto del resultado deseado es verdadero y deduciendo luego de esta suposición un resultado que contradiga una verdad ya conocida. Esto significa que lo opuesto del resultado deseado no es verdadero; por tanto, también el resultado deseado debe ser cierto.

Una demostración por el principio de inducción procede de la manera siguiente. Se afirma que una proposición $P(i)$ es verdadera para todos los enteros i . Para comprobar la afirmación, se requieren dos cosas:

- Demostrar que la afirmación es verdadera para algún entero pequeño, digamos $i = 1$.
- Suponer que es verdadera para cualquier entero arbitrario k . Y mostrar entonces que debe, por tanto, ser verdadera para el siguiente entero $k + 1$.

El último paso significa que ya que el resultado es verdadero para $i = 1$, debe serlo para el siguiente entero $i = 2(1 + 1)$; entonces debe ser cierto para $i = 3(2 + 1)$; y así sucesivamente para todos los otros enteros.

Otro método general de prueba, el cual se explica en la siguiente subsección, resulta especialmente válido en un álgebra booleana con sólo dos elementos.

Advierta que el símbolo para la multiplicación lógica (•) se omite con frecuencia por simplicidad, y que $x \cdot y$ se escribe como xy . Sin embargo, siempre que pudiera haber confusión, el símbolo del operador debe mostrarse en forma explícita. La confusión puede surgir, por ejemplo, si el nombre de una variable lógica consiste en dos caracteres múltiples. Así, una variable podría denominarse OUT2, designando el número de salida 2, en vez del producto lógico de O y U y T y 2. En este capítulo se asignan nombres simples a las variables; por consiguiente, a menudo omitiremos el símbolo del producto lógico. Lo mostraremos explícitamente cuando sea necesario, para evitar confusiones. De la misma manera, cuando los nombres de las variables consten de más de un símbolo, éstos se encerrarán entre paréntesis. Así, los paréntesis en (OUT2)(OUT3) permiten la omisión del símbolo de multiplicación lógica. A continuación abordamos los teoremas.

Teorema 1 Ley de elementos nulos

$$1a. x + 1 = 1$$

$$2b. x \cdot 0 = 0$$

Advierta que cada una de estas leyes se desprende de otra por dualidad; en consecuencia, sólo una necesita demostración explícita. Vamos a demostrar la segunda.

$$\begin{aligned}
 x \cdot 0 &= 0 + (x \cdot 0) && \text{Postulado 2a} \\
 &= (x \cdot x') + (x \cdot 0) && \text{Postulado 5b} \\
 &= x \cdot (x' + 0) && \text{Postulado 4a} \\
 &= x \cdot x' && \text{Postulado 2a} \\
 &= 0 && \text{Postulado 5b}
 \end{aligned}$$

El teorema 1a se cumple por dualidad. ■

Teorema 2 involución $(x')' = x$

En palabras, éste establece que el complemento del complemento de un elemento es ese elemento mismo. Esto resulta de la observación de que el complemento de un elemento es único. Los detalles de la demostración se dejan al lector (véase el problema 1d). ■

Teorema 3 idempotencia

$$3a. x + x = x$$

$$3b. x \cdot x = x$$

Para demostrar el teorema 3a,

$$\begin{aligned}
 x + x &= (x + x) \cdot 1 && \text{Postulado 2b} \\
 &= (x + x) \cdot (x + x') && \text{Postulado 5a} \\
 &= x + x \cdot x' && \text{Postulado 4b} \\
 &= x + 0 && \text{Postulado 5b} \\
 &= x && \text{Postulado 2a}
 \end{aligned}$$

El teorema 3b es verdadero por dualidad. ■

Teorema 4 absorción

$$4a. x + xy = x$$

$$4b. x(x + y) = x$$

Para demostrar el teorema 4a,

$$\begin{aligned}
 x + x \cdot y &= x \cdot 1 + x \cdot y && \text{Postulado 2b} \\
 &= x \cdot (1 + y) && \text{Postulado 4a} \\
 &= x \cdot 1 && \text{Postulado 3a y teorema 1a} \\
 &= x && \text{Postulado 2b}
 \end{aligned}$$

El teorema 4b es verdadero por dualidad. ■

Teorema 5 simplificación

$$5a. x + x'y = x + y$$

$$5b. x(x' + y) = xy$$

Para demostrar el teorema 5b,

$$\begin{aligned}
 x(x' + y) &= xx' + xy && \text{Postulado 4a} \\
 &= 0 + xy && \text{Postulado 5b} \\
 &= xy && \text{Postulado 2a}
 \end{aligned}$$

El teorema 5a es verdadero por dualidad. ■

Teorema 6 ley asociativa

$$6a. x + (y + z) = (x + y) + z = x + y + z$$

$$6b. x(yz) = (xy)z = xyz$$

La demostración del teorema 6a requiere un poco de ingenuidad. En primer lugar, a partir del producto lógico de los dos lados de la primera igualdad:

$$A = [x + (y + z)] \cdot [(x + y) + z]$$

Se desarrolla este producto utilizando la ley distributiva, considerando primero la cantidad dentro de los primeros corchetes como una unidad al empezar, y se prosigue a partir de ahí; después se trata la cantidad en el segundo corchete como una unidad al empezar, y de ahí se continúa. El resultado es $A = x + (y + z)$ en el primer caso, y $A = (x + y) + z$ en el segundo caso. (Trabaje los detalles.) El resultado se deduce por transitividad (si cada una de dos cantidades son iguales a una tercera, éstas deben ser iguales entre sí). Puesto que el resultado es el mismo sin importar qué variables individuales se agrupan en paréntesis, éstos no son necesarios y es posible eliminarlos.

El teorema 6b se cumple por dualidad. ■

Teorema 7 consenso

$$7a. xy + x'z + yz = xy + x'z$$

$$7b. (x + y)(x' + z)(y + z) = (x + y)(x' + z)$$

Para demostrar el teorema 7a,

$$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + yzx + yzx' \\ &= (xy + xyz) + (x'z + x'zy) \\ &= xy + x'z \end{aligned}$$

Postulado 5a

Postulado 4a

Postulado 3b y teorema 6a

Teorema 4a

El teorema 7b es verdadero por dualidad. ■

Teorema 8 ley de De Morgan

$$8a. (x + y)' = x'y'$$

$$8b. (xy)' = x' + y'$$

Se demuestra el teorema 8a comprobando que $x'y'$ satisface ambas condiciones en el postulado 5 relativas a ser el complemento de $x + y$.

Condición 1

$$\begin{aligned} (x + y) + x'y' &= (x + x'y') + y \\ &= (x + y') + y \\ &= x + (y' + y) \\ &= x + 1 \\ &= 1 \end{aligned}$$

Postulado 3a y teorema 6a

Teorema 5a

Teorema 6a

Postulado 5a

Teorema 1a

Condición 2

$$\begin{aligned} (x + y)(x'y') &= xx'y' + yx'y' \\ &= 0 \cdot y' + x'(yy') \\ &= 0 \end{aligned}$$

Postulado 3b y 4a

Postulado 5b y 3b

Postulado 5b y teorema 1b

El teorema 8b es verdadero por dualidad. ■

La confirmación de otros resultados importantes y del conjunto de problemas se deja al lector, los usaremos aquí como si ya se hubieran demostrado. Éstos incluyen los siguientes:

1. Los elementos identidad 0 y 1 son elementos distintos.
2. Los elementos identidad son únicos.
3. El inverso de un elemento es único.

Ejercicio 1. Demuestre que cada elemento identidad es el complemento del otro. ♦

Álgebra de conmutación

En el álgebra booleana analizada hasta ahora en este libro, el dominio no se ha restringido. Esto es, no se ha impuesto limitación sobre el número de elementos en el álgebra booleana. De acuerdo con los postulados de Huntington, sabemos que en toda álgebra booleana hay dos elementos específicos: los elementos identidad. Por consiguiente, cualquier álgebra booleana cuenta *por lo menos* con dos elementos. En este libro, nos vamos a limitar de aquí en adelante al álgebra booleana de *dos elementos*.³

En un artículo de 1937, Claude Shannon puso en práctica un álgebra booleana de dos elementos con un circuito de interruptores.⁴ Ahora bien, un interruptor es un dispositivo que puede ubicarse en una de dos posiciones estables: *desactivado* o *activado*; igualmente es posible denominar estas posiciones 0 y 1 (o lo inverso). Por esta razón, el álgebra booleana de dos elementos se ha denominado *álgebra de conmutación*, y los propios elementos identidad reciben el nombre de *constantes de conmutación*. De manera similar cualesquiera variables que representen a las constantes de conmutación se conocen como *variables de conmutación*.

Esto explica parte de la terminología común que se utiliza en esta área, aunque ya hemos utilizado cierta terminología cuya fuente no es evidente. Los términos *multiplicación lógica* y *adición lógica* se presentaron en el primero de los postulados de Huntington. Para explicar de dónde proviene el adjetivo *lógica*, tendremos que recurrir a una ligera digresión.

Durante siglos se han formulado varios sistemas algebraicos distintos, en diferentes contextos. El lenguaje que se usa al describir cada sistema y las operaciones que se efectúan en éste tienen sentido en el contexto en el que se desarrolló el álgebra. El álgebra de *conjuntos* es uno de estos casos; otro es un sistema llamado *lógica de proposiciones*, el cual se generó en el estudio de la filosofía.

Es posible que diferentes sistemas algebraicos, provenientes de contextos distintos, cuenten con propiedades similares. Esta posibilidad constituye la base de la siguiente definición.

Se dice que dos sistemas algebraicos son isomórficos si pueden hacerse idénticos cambiando los nombres de los elementos y los nombres y símbolos utilizados para designar las operaciones.

La lógica de proposiciones tiene que ver con proposiciones simples, sean o no verdaderas o falsas, con la manera en que es posible combinar las proposiciones simples con proposiciones más complejas, y cómo puede deducirse la veracidad o falsedad de las proposiciones complejas a partir de lo cierto o falso de las simples. Una proposición simple es un enunciado aseverativo que quizá sea verdadero o falso, pero no ambas cosas. Se dice que tiene dos *valores de verdad* posibles: verdadero (V) o falso (F). Son ejemplos

“La tierra es plana”. F

“La suma de dos enteros positivos es positiva”. V

³ Es posible demostrar que el número de elementos en cualquier álgebra booleana es alguna potencia de $2:2^n$, para $n \geq 1$.

⁴ Implementar una expresión matemática significa construir un modelo de un sistema físico, o el sistema físico mismo, cuyo desempeño iguale el resultado de la operación matemática. Otro verbo con el mismo significado es “realizar”. El sistema físico, o su modelo, obtenido de esa manera se dice que será una *implementación* o una *realización*.

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

Figura 1. Tabla de verdad AND.

No es la intención aquí seguir con detalles este tema. Sin embargo, ocurre que el álgebra booleana de dos valores es isomórfica con la lógica de proposiciones. En consecuencia, cualesquiera que sean la terminología, operaciones y técnicas utilizadas en lógica pueden aplicarse al álgebra booleana, y viceversa.

Como ejemplo, los elementos del álgebra booleana (1 y 0) corresponden a la verdad (V) o falsedad (F) de proposiciones; V y F podrían indicarse mediante 1 y 0, respectivamente, o lo opuesto. O sería posible que los elementos del álgebra booleana, 1 y 0, se llamen "valores de verdad", aunque las ideas de verdad y falsedad no tienen significado filosófico en el álgebra booleana.

Se dice que una proposición será la *negación* de otra proposición si es falsa siempre que la otra es verdadera. ("No neva" es la negación de "está nevando".) Si p es una proposición, entonces $\neg p$ es su negación. Ésta es isomórfica con el complemento en álgebra booleana, y el mismo símbolo (prima) puede utilizarse para representarla: $\neg p$ se escribe p' . Podremos usar el término *negación* en el álgebra booleana para que represente al *complemento*.

Existen relaciones isomórficas similares entre las operaciones del álgebra booleana y las conjunciones que unen proposiciones entre sí. Sin embargo, una consideración adicional de lo anterior se verá en la sección siguiente.

2 OPERACIONES DE CONMUTACIÓN

Una operación unaria y dos operaciones binarias, con nombres tomados de la lógica de proposiciones, se introdujo en los postulados de Huntington. En el álgebra de dos elementos (de conmutación) es común renombrar estas operaciones, recurriendo también en este caso a términos que provienen de la lógica.

La operación AND

Vamos a considerar primero la multiplicación lógica (AND) de dos variables, xy . La operación producirá diferentes valores dependiendo de los que tomen cada uno de los elementos que representan las variables. Así, si $x = 1$, entonces del postulado 2b, $xy = y$; pero si $x = 0$, entonces del teorema 1, $xy = 0$, independientemente de y . Estos resultados puede presentarse en una tabla (figura 1) que lista todas las combinaciones posibles de valores de x e y y los valores correspondientes de xy .

Esta tabla se denomina *tabla de verdad*. Ni la palabra *verdad* ni el nombre de la operación, AND, tienen sentido en términos del álgebra booleana; los términos se toman de la lógica de proposiciones. La operación xy es similar a la proposición compuesta

"La luna está llena (x) y la noche es joven (y)."

Esta proposición compuesta es verdadera sólo si *ambas* de las proposiciones simples "la luna está llena" y "la noche es joven" son verdaderas; es falsa en todos los demás casos. De tal manera, xy en la tabla de verdad es 1 sólo si tanto x como y son 1. Estudie la tabla ampliamente.

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Figura 2. Tabla de verdad OR.

La operación OR

Además de “y”, la conjunción “o” es otra forma de conectar dos proposiciones. Sin embargo, esta conjunción presenta cierta ambigüedad. Supóngase que se afirma que a las 6 en punto lloverá o nevará. La proposición compuesta será verdadera si llueve, si neva o si *llueve y neva* —esto es, si cualquiera o ambas de las proposiciones simples son verdaderas. El único caso en que será falsa es si no llueve ni neva.⁵ Estos resultados, al igual que la tabla de verdad para AND, se pueden resumir en una tabla de verdad para la conjunción OR. (Confirme las entradas en la figura 2.)

Veamos cómo se compara esto en el álgebra de conmutación con la adición lógica, $x + y$.

A partir del postulado 2a de Huntington, si $y = 0$, entonces $x + y = x$, por lo que $x + y$ corresponderá a cualquier valor que tenga x . Pero si $y = 1$, entonces a partir del teorema 1a, $x + y = x + 1 = 1$, para ambos valores de x . Verifique que estos valores corresponden exactamente a los valores que se dan en la tabla de verdad OR.

La operación NOT

Por último, la operación complemento es isomórfica con la negación, o NOT, en lógica. Resulta un asunto sencillo establecer una tabla de verdad para esta operación; dejaremos que el lector la realice.

Ejercicio 2. Construya una tabla de verdad para el operador NOT.

Comentario

El isomorfismo del álgebra de conmutación con la lógica de proposiciones ha presentado una nueva herramienta, la tabla de verdad, que se usa para establecer relaciones entre operaciones de conmutación. Por ejemplo, los teoremas demostrados mediante la aplicación de los postulados de Huntington junto con los teoremas probados pueden demostrarse también a partir de las tablas de verdad. Ilustraremos lo anterior aplicando las tablas de verdad de las operaciones AND, OR y NOT para demostrar la validez de la primera forma de la ley de De Morgan:

$$(x + y)' = x'y'$$

El resultado se presenta en la figura 3. Las últimas dos columnas son las mismas para todas las combinaciones posibles de las variables de conmutación; esto demuestra la ley de De Morgan.

El procedimiento que acaba de utilizarse para establecer la ley de De Morgan ilustra un método general para comprobar resultados en el álgebra de conmutación.

Al método de demostración que se basa en tablas de verdad para probar una relación entre variables de conmutación, que verifica que la relación es verdadera para todas las combinaciones posibles de valores de las variables, se le denomina método de inducción perfecta.

⁵ La ambigüedad en la conjunción “o” en un enunciado tal como “ese animal es un gato o un perro”. En este caso, es posible que el animal sea un gato o un perro, pero ciertamente no ambos. En tal caso, algo más que la suma lógica se necesita para describir la conjunción. Esto se analizará en la sección 5.

x	y	x'	y'	$x + y$	$(x + y)'$	$x'y'$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Figura 3. Tabla de verdad para la ley de De Morgan.

Es posible que este enfoque exhaustivo se vuelva problemático si es grande el número de variables. Probablemente se considere este enfoque intelectual y estéticamente menos satisfactorio que aplicar los postulados del álgebra booleana y los teoremas ya demostrados. Aun así, éste resulta válido.

En la ley de De Morgan, dejemos que cada lado se denomine z . Entonces $z = (x + y)'$ y $z = x'y'$. Existen dos operaciones, OR y NOT, en el lado derecho de la primera expresión. Estimulados por los libros de texto que quizá haya consultado en la biblioteca, algunos estudiantes estarían tentados a pensar que la manera de evaluar el lado derecho implica únicamente imaginar cuál de los dos operadores "se aplica primero". Para aclarar el asunto, debe ser suficiente concentrarse en la importancia de los paréntesis en las expresiones algebraicas (tanto booleanas como ordinarias) y en los significados de cada operador booleano.

Un conjunto de paréntesis es una herramienta que se utiliza para agrupar algunas variables en una unidad; las operaciones se van a efectuar sobre la unidad, no en partes de ella dentro del paréntesis. Así $(x + y)'$ significa formar la unidad $x + y$ y luego tomar el complemento de esta unidad. A la unidad se le puede asignar un nombre, por ejemplo w . De esa manera $(x + y)'$ significa w' . En términos de w , ¿no se podría formular la pregunta de cuál operador ocurre primero y debe aplicarse antes que el otro? Obviamente efectuará la operación OR de x e y , para obtener w ; y después realizará la operación NOT w .

De manera similar, para $z = x'y'$, la pregunta, "¿qué hago primero, NOT y luego AND, AND primero y luego NOT?" no tiene sentido. Quizá resulte más simple si definimos $u = x'$ y $v = y'$, entonces $z = uv$. Ahora ya no hay pregunta, efectuamos la operación AND de las dos cosas, u y v , que viene a ser la NOT de x y la NOT de y , respectivamente. Por tanto, $z = (\text{NOT } x) \text{ AND } (\text{NOT } y)$; esto no podría ser más claro.

Es una práctica común, sólo por simplicidad, omitir los paréntesis alrededor de las variables a las que se aplica la operación AND con el entendimiento de que dicha operación se efectuará antes que otras operaciones sobre la unidad a la que se aplica AND. En realidad, la misma convención —que la operación multiplicación se efectúa antes de la suma— se usa también en el álgebra ordinaria. De manera que en lugar de tratar de memorizar el orden en el que se van a efectuar las operaciones en una expresión determinada, podrá concentrarse en el significado de las operaciones fundamentales AND, OR y NOT. De ese modo no se equivocará.

3 EXPRESIONES DE CONMUTACIÓN

Volvamos a la ley de De Morgan en el teorema 8. Cada lado consiste en ciertas variables de conmutación relacionadas por medio de las operaciones AND, OR y NOT. Cada una constituye un ejemplo de una *expresión de conmutación*, la cual definimos ahora formalmente:

Una expresión de conmutación es una relación finita entre variables de conmutación (y posiblemente en las constantes de conmutación 0 y 1), relacionadas por las operaciones AND, OR y NOT.

Algunos ejemplos simples de expresiones de conmutación son $xx' + x$, $z(x + y)'$ e $y + 1$. Un ejemplo más complejo de una expresión de conmutación es

$$E = (x + yz)(x + y') + (x + y)'$$

donde E significa "expresión".

Advierta que las expresiones están conformadas por variables, o sus complementos, sobre las cuales se van a efectuar diversas operaciones. Por simplicidad, nos referiremos a las variables o complementos de variables como *literales*. La expresión E consiste en el producto lógico de dos expresiones sumadas lógicamente a otro término. (En adelante, cuando analicemos sumas y productos lógicos, omitiremos el adjetivo *lógicos*. Sin embargo, recuérdese que éste siempre está implicado.) El segundo término en el producto es en sí mismo una suma de literales, $x + y'$. El primer término en el producto no puede describirse simplemente como una suma o un producto.

Una expresión dada puede ponerse en muchas formas equivalentes aplicando las *leyes booleanas* (esto es lo que llamamos en forma resumida los postulados y teoremas). No obstante, posiblemente usted pregunte, ¿de qué se trata? ¿Por qué molestarse en realizar gran cantidad de álgebra para obtener una forma diferente? Esta vez brindaremos únicamente una respuesta tentativa e incompleta. Cada variable de conmutación en una expresión representa presumiblemente una señal; las operaciones lógicas se pondrán en práctica (se efectuarán) por medio de unidades de hardware cuya salida total corresponderá a la expresión considerada. Si es posible representar una expresión determinada en formas diferentes, entonces pueden utilizarse diferentes combinaciones de hardware para conseguir el mismo resultado total. Presumiblemente, algunas configuraciones de hardware tienen ventajas sobre otras. En el capítulo 3 se abordarán métodos más sistemáticos para tratar diferentes representaciones de expresiones de conmutación; su implementación se continuará en el capítulo 4. Aquí solamente planteamos el escenario.

Volvemos ahora a la expresión E , renombrada E_1 en lo que sigue. Se encuentran expresiones equivalentes aplicando leyes específicas del álgebra de conmutación. La aplicación de la ley distributiva al término producto y la ley de De Morgan al último término conducen a E_2 ; entonces

$$\begin{aligned} E_1 &= (x + yz)(x + y') + (x + y)' \\ E_2 &= xx + xy' + xyz + y'yz + x'y' \\ E_3 &= x + x(y' + yz) + x'y' && \text{Teorema 3a, postulado 4a y 5b} \\ E_4 &= x + x'y' && \text{Postulado 4a y teorema 4a} \\ E_5 &= x + y' && \text{Teorema 5a} \end{aligned}$$

Una expresión bastante complicada se ha reducido a una muy simple. Advierta que E_2 contiene un término yy' , que es igual al elemento identidad 0. Afirmamos que la expresión es *redundante*. Mas generalmente, una expresión será redundante si contiene:

- Literales repetidas (xx o $x + x$);
- una variable y su complemento (xx' o $x + x'$);
- constantes de conmutación mostradas explícitamente (0 o 1).

Las redundancias en las expresiones no se necesitan poner en práctica en el hardware; pueden eliminarse de las expresiones en las cuales aparece.

Minitérminos, maxitérminos y formas canónicas

Dada una expresión que depende de n variables, hay dos formas específicas y únicas en las cuales la expresión se puede convertir. Estas formas son el tema de esta sección.

La expresión E_1 en la sección precedente tuvo mezclas de términos que eran productos o sumas de otros términos. Además, aunque E_1 en apariencia dependía de tres variables, una de estas variables era redundante. La forma final equivalente correspondía a la suma de dos términos, siendo cada uno una literal sencilla.

En el caso general, las expresiones dependen de n variables. Consideraremos dos casos no redundantes. En uno de ellos, una expresión está compuesta sólo por sumas de términos, y cada término se integra mediante un producto de literales. Naturalmente, esta expresión se denominaría una forma de *suma de productos* (s de p). El número máximo de literales en un producto no redundante es n . En el segundo caso que se va a considerar, una expresión consta únicamente de un producto de términos, y cada término está conformado por una suma de literales; ésta es la forma de *producto de sumas* (p de s). De nuevo, el número máximo de literales en una suma no redundante es n .

Suponga que un término producto en una expresión de suma de productos, o un término suma en una forma de producto de sumas, tiene menos literales que el número máximo n . Para distinguir casos de este tipo de uno en el cual cada término está "lleno", establecemos la siguiente definición:

Una expresión de suma de productos o de producto de sumas dependiente de n variables es canónica si contiene literales no redundantes y cada producto o suma tiene exactamente n literales.⁶

Cada producto o término suma en una expresión canónica tiene tantas literales como el número de variables.

EJEMPLO 1

Un ejemplo de una expresión que tiene tres variables, en forma de producto de sumas, es la E_1 siguiente (no la E_1 anterior). Ésta se convierte en una forma de suma de productos como se señala, con cada paso justificado por las leyes de conmutación listadas.

$$\begin{aligned} E_1 &= (x' + y' + z)(x + y + z')(x + y + z) \\ E_2 &= (x' + y' + z)[(x + y)(x + y) + (x + y)(z + z') + zz'] && \text{Postulado 4a} \\ E_3 &= (x' + y' + z)(x + y) && \text{Postulado 1 y teorema 3b} \\ E_4 &= xy' + x'y + xz + yz && \text{Postulado 4a y 5b, teorema 7} \end{aligned}$$

Al ir de E_1 a E_4 , se eliminaron las redundancias en cada paso. La expresión original está en la forma de producto de sumas, con el número máximo de literales en cada término; en consecuencia, es canónica. La forma final, por otra parte, es la forma de suma de productos, pero no es canónica; ésta sólo cuenta con dos literales en cada término.

Dada una expresión no canónica de suma de productos, siempre es posible convertirla a la forma canónica. —¡Si existe alguna razón para hacerlo! El término xy' en la expresión en la E_4 , por ejemplo, carece de la variable z . Por consiguiente, multiplique el término por $z + z'$, que es igual a 1 y por ello no cambia el valor lógico; después se desarrolla. La misma idea puede utilizarse con los otros términos. Al efectuar estos pasos sobre E_4 se llega a lo siguiente:

$$\begin{aligned} E_5 &= xy'(z + z') + x'y(z + z') + xz(y + y') + yz(x + x') \\ E_6 &= xy'z + xy'z' + x'yz + x'yz' + xyz \end{aligned}$$

(Debe confirmar la última línea; advierta que se han eliminado las redundancias que se crean en la primera línea al aplicar el postulado 4a.) Ésta se encuentra ahora en la forma canónica de suma de productos. ■

⁶ Algunos autores utilizan *canonical* en lugar de *canónica*.

En una expresión de suma de productos dependiente de n variables, con el fin de distinguir entre los términos producto que tiene n literales (el máximo) y aquéllos con un número menor que n , se establece la siguiente definición:

Un producto de literales no redundante canónico recibe el nombre de minitérmino.⁷

Esto es, un minitérmino es un producto no redundante de tantas literales como variables haya en una expresión determinada. Así, cada término en E_6 es un minitérmino, y por ello la expresión completa es una suma de minitérminos.

La misma idea se aplica a una expresión de producto de sumas. Para distinguir entre términos producto que tengan el número máximo de literales y otros, se hace la siguiente definición:

Una suma de literales no redundante canónica se denomina un maxitérmino.

Cada factor en la expresión E_1 en el desarrollo anterior es un maxitérmino; la expresión completa constituye un producto de maxitérminos. Si una expresión de producto de sumas no es inicialmente canónica, es posible convertirla a esa forma de una manera similar a la que se usó para el caso de la suma de productos pero con las operaciones intercambiadas.

Ejercicio 3. Convierta la siguiente expresión a un producto canónico de maxitérminos: $E = (x + y')(y' + z')$.

Respuesta⁸

Generalización de la ley de De Morgan

Una de las leyes booleanas que ha encontrado una amplia aplicación es la ley de De Morgan. Ésta se estableció primero como el teorema 8 en términos de dos variables. Repetiremos ésta y su forma dual, donde las operaciones suma y producto se intercambian:

$$(a) (x_1 + x_2)' = x_1'x_2' \quad y \quad (b) (x_1x_2)' = x_1' + x_2'$$

En palabras, el complemento de la suma (producto) de dos variables de conmutación produce el mismo resultado que multiplicar (sumar) sus complementos. Supóngase que vamos a incrementar el número de variables a tres, ¿el resultado seguiría siendo verdadero? Esto es, deseamos efectuar la siguiente operación: $(A + B + C)'$. Vamos a renombrar $A + B$ como D ; entonces lo que queremos es $(D + C)'$. Pero esto es exactamente $D'C'$, por la ley de De Morgan para dos variables; y de nuevo por la ley de De Morgan, $D' = (A + B)' = A'B'$. Por consiguiente, el resultado final es

$$(a) (A + B + C)' = A'B'C' \quad (b) (ABC)' = A' + B' + C' \quad (1)$$

(La segunda forma se obtiene por dualidad.)

¿Por qué detenemos en tres variables? El caso general se escribe del modo siguiente:

$$(x_1 + x_2 + \dots + x_n)' = x_1'x_2' \dots x_n' \quad (2)$$

$$(x_1x_2x_3 \dots x_n)' = x_1' + x_2' + \dots + x_n' \quad (3)$$

En palabras, (2) señala que el complemento de la suma lógica de cualquier número de variables de conmutación es igual al producto lógico del complemento de esas variables. (En términos diferentes, el complemento de una secuencia de sumas es igual al producto del complemento de

⁷ El nombre no parece tener sentido, ¿qué significado tiene "min"? Si tiene alguno, a partir del hecho de que tenemos que pasar de términos con dos literales a algunos con tres literales, ¿se pensaría que sería necesario utilizar el nombre "max"? Su asombro tendrá que persistir hasta el capítulo 3, cuando todo este asunto se aclare.

⁸ $E = (x + y' + z)(x + y' + z')(x' + y' + z')$

las variables en esa secuencia.) En (3), intercambiamos las operaciones producto y suma. Escribir únicamente la generalización no hace que ésta sea verdadera; se le pide al lector que lo demuestre.

Ejercicio 4. Demuestre una de las leyes generalizadas de De Morgan por medio de inducción matemática. Esto es, suponga que es verdadera para k variables, y muestre que es válida para $k + 1$. Puesto que sabemos que es verdadera para dos variables, entonces será verdadera para tres (como ya se demostró); puesto que es verdadera para tres, entonces será verdadera para 4 y así sucesivamente. ♦

Es posible concluir algo más general a partir de la ley de De Morgan. En (2) y (3), la generalización implica incrementar el número de variables en el teorema. En los lados de la izquierda, las operaciones en las que se está afectando el complemento son la suma y el producto. Suponga ahora que estas operaciones son las que se generalizarán. Esto es, a la izquierda, vamos a tomar el complemento de alguna expresión que depende de n variables, donde las operaciones $+$ y \cdot se efectúan en diversas combinaciones. ¿Se producirá el mismo resultado si tomamos la misma expresión pero intercambiamos las operaciones de suma y producto mientras se complementan todas las variables? La generalización es:

$$E'(x_1, x_2, \dots, x_n, +, \cdot) = E(x_1', x_2', \dots, x_n', \cdot, +) \quad (4)$$

Advierta el orden de las operaciones de suma y producto en cada lado; ello indica que se intercambian las dos operaciones en los dos lados, siempre que éstas aparecen. El resultado puede demostrarse (no lo haremos aquí) mediante inducción matemática sobre el número de operaciones. (Puede hacerlo usted, si lo desea.)

EJEMPLO 2

Se da la siguiente expresión: $E = xy'z + x'y'z' + x'yz$. Para encontrar el complemento de E , intercambiamos las operaciones $+$ y \cdot y sustituimos cada variable por su complemento. De ese modo,

$$E'(x + y + z')(x + y + z)(x + y' + z')$$

Reescribiremos primero esta expresión en la forma de suma de productos efectuando las operaciones producto sobre los términos dentro del paréntesis y utilizando el álgebra booleana necesaria para simplificar. (Efectúe estos pasos antes de verificar su trabajo en lo que sigue.)

$$\begin{aligned} E' &= (x'y + x'z + yx + y + yz + z'x + z'y)(x + y' + z') \\ &= (x'z + xz' + y)(x + y' + z') \\ &= xy + yz' + x'y'z + xz' \end{aligned}$$

Para confirmar que esta expresión produce correctamente el complemento de E , puede usted tomar su complemento utilizando la misma generalización de la ley de De Morgan y ver si se produce la expresión original correspondiente a E . ■

Ejercicio 5. Tome el complemento de E' en la expresión anterior, utilizando (4). Ponga el resultado en la forma de suma de productos y verifique que el resultado es la misma E como en el ejemplo 2. ♦

x	y	x'	y'	$x'y'$	E_1	E_2
					$x + x'y'$	$x + y'$
0	0	1	1	1	1	1
0	1	1	0	0	0	0
1	0	0	1	0	1	1
1	1	0	0	0	1	1

Figura 4. Tabla de verdad para E_1 y E_2 .

4 FUNCIONES DE CONMUTACIÓN

En la sección anterior observamos varios ejemplos en los cuales una expresión de conmutación determinada se convirtió en otras expresiones equivalentes aplicándole leyes booleanas. Las expresiones equivalentes tienen los mismos valores lógicos para todas las combinaciones de valores de las variables. El concepto de "función" desempeña un papel muy importante en el álgebra ordinaria. Hasta ahora, dicho concepto no se había presentado en el álgebra de conmutación. Lo haremos ahora.

La explicación considera dos expresiones simples:

$$E_1 = x + x'y' \quad \text{y} \quad E_2 = x + y'$$

Cada expresión depende de dos variables. Colectivamente, las variables pueden tomar $2^2 = 4$ combinaciones de valores. (Para n variables, el número de combinaciones de valores es 2^n .) Para cualquier combinación de valores de variables, cada expresión toma un valor que se encuentra sustituyendo los valores de las variables en él. Cuando esto se efectúa para todas las combinaciones de valores de variables, el resultado es la tabla de verdad de la figura 4. (Confirme todas las entradas.) Las últimas dos columnas son iguales. Esto difícilmente sorprende dado el teorema 5a. (Revíselo.)

El ejemplo anterior ilustra el principio de que diferentes expresiones pueden conducir a los mismos valores de verdad para todas las combinaciones de los mismos valores de variables.

Ejercicio 6. Recurriendo a una tabla de verdad, confirme que la expresión: $E = xy + xy' + y'$ tiene los mismos valores de verdad que E_1 y E_2 en la figura 4. ♦

Debe haber algo más fundamental que las expresiones equivalentes; lo cual puede identificarse mediante sus valores de verdad. Con esta base, definimos una función de conmutación de la manera siguiente:

Una función de conmutación es una asignación específica de valores de conmutación 0 y 1 para todas las combinaciones posibles de valores que toman las variables de las cuales depende la función.

En el caso de una función de n variables existen 2^n combinaciones posibles de valores. En cada combinación de valores, la función toma uno de dos valores. En consecuencia, el número de asignaciones distintas de dos valores para 2^n cosas es 2 a la potencia 2 a la n .

El número de funciones de conmutación de n variables es 2 a la 2^n .

De acuerdo con esto, existen 16 funciones de conmutación de dos variables y 256 funciones de tres variables; el número asciende rápidamente para más variables.

Considere ahora las funciones $f_1(x, y, z)$ y $f_2(x, y, z)$ cuyos valores de verdad se presentan en la figura 5. Cualquier operación de conmutación (AND, OR, NOT) puede efectuarse sobre estas funciones y computarse los resultados utilizando la tabla de verdad. De tal manera, es posible que f_1' se forme asignando el valor 1 siempre que f_1 tenga el valor 0, y viceversa. Otras com-

x	y	z	f_1	f_2	f_1'	f_2'	$f_1 + f_2'$	$(f_1 f_2)'$
0	0	0	0	1	1	0	0	1
0	0	1	1	0	0	1	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	1	0	0	1	0
1	0	0	0	1	1	0	0	1
1	0	1	0	1	1	0	0	1
1	1	0	1	1	0	0	1	0
1	1	1	1	1	0	0	1	0

Figura 5. Tablas de verdad para varias funciones.

binaciones de f_1 , f_2 o sus complementos se forman directamente a partir de la tabla de verdad. Como ejemplos, también se presentan $f_1 + f_2'$ y $(f_1 f_2)'$.

Operaciones de conmutación en funciones de conmutación

Resulta claro que las funciones —y, en consecuencia, las expresiones que representan funciones— pueden tratarse como si fueran variables.

De tal manera que las leyes de conmutación se aplican igualmente bien a las expresiones de conmutación como a las variables que representan los elementos de conmutación.

Advierta que hay una diferencia en significado entre una *función* de conmutación y una *expresión* de conmutación. Una función se define listando sus valores de verdad para todas las combinaciones de valores de las variables, esto es, mediante su tabla de verdad. Una expresión, en cambio, es una combinación de literales vinculadas mediante operaciones de conmutación. Para una combinación determinada de valores de variables, la expresión tomará un valor de verdad.

Si los valores de verdad tomados por una expresión, para todas las combinaciones posibles de valores de las variables, son los mismos que los correspondientes valores de verdad de la función, afirmamos en ese caso que la expresión *representa* la función. Como se observó antes, es posible escribir más de una expresión para representar una función específica. Así, lo que es fundamental es la *función* de conmutación. De todas las expresiones que pueden representar una función, podríamos buscar las particulares que de alguna manera ofrezcan una ventaja sobre las otras. Este asunto se tratará de manera adicional en el capítulo 3.

Código decimal	x	y	z	f	Minterms	Maxterms
0	0	0	0	0	$x'y'z'$	$x + y + z$
1	0	0	1	0	$x'y'z$	$x + y + z'$
2	0	1	0	1	$x'yz'$	$x + y' + z$
3	0	1	1	1	$x'yz$	$x + y' + z'$
4	1	0	0	1	$xy'z'$	$x' + y + z$
5	1	0	1	1	$xy'z$	$x' + y + z'$
6	1	1	0	0	xyz'	$x' + y' + z$
7	1	1	1	1	xyz	$x' + y' + z'$

Figura 6. Tabla de verdad para la función de ejemplo.

Número de términos en formas canónicas

Al tratar como equivalentes algunas expresiones de conmutación en la sección anterior, en el ejemplo 1 se incluyen varias expresiones diferentes dependientes de tres variables.

Afirmamos que todas estas expresiones representan la misma función. E_1 , en forma canónica de producto de sumas, tiene tres términos como factores en el producto, esto es, tres maxitérminos. Por otra parte, E_6 está en la forma de suma de productos y tiene cinco términos en la suma, esto es cinco minitérminos. La suma del número de minitérminos y maxitérminos es igual a 8, que es 2^3 , el número de combinaciones posibles de 3 bits.

Ésta es una observación con la que vamos a seguir construyendo una tabla de verdad de la función representada por las expresiones E_1 y E_6 . Escribimos primero los minitérminos y los maxitérminos en el orden xyz : 000 a 111.

$$E_6 = x'yz' + x'yz + xy'z' + xy'z + xyz$$

$$E_1 = (x + y + z)(x + y + z')(x' + y' + z)$$

La tabla se presenta en la figura 6. Es posible advertir algo muy interesante al examinar las expresiones para los minitérminos y los maxitérminos. Suponga que tomamos el complemento del minitérmino correspondiente a 000. Mediante la ley de De Morgan, éste es exactamente el primer maxitérmino. Confirme que lo anterior es en realidad verdadero para cada renglón en la tabla. A partir de esto concluimos que la determinación de la forma canónica del producto de sumas consiste en *aplicar la ley de De Morgan a cada minitérmino que no está presente*.

El resultado en este ejemplo es uno general. Dada la forma canónica de la suma de productos de una función de conmutación, la manera de obtener la expresión canónica del producto de sumas es la siguiente:

- Se aplica la ley de De Morgan al complemento de cada minitérmino que está *ausente* en la expresión de suma de productos.
- Luego se forma el producto de los maxitérminos resultantes.

De manera inversa, para obtener la expresión de la suma de productos a partir de una expresión determinada de productos de sumas,

- Se aplica la ley de De Morgan a cada término suma ausente del producto.
- Luego se forma la suma de los minitérminos resultantes.

(Usted puede confirmar lo anterior a partir de la misma tabla de verdad.)

Ejercicio 7. Dada la expresión del producto de sumas en E_1 en esta sección, empleamos el enfoque anterior para determinar la forma correspondiente de suma de productos. ♦

Este enfoque evita la necesidad de efectuar álgebra de conmutación para convertir de una forma a la otra; en vez de eso, dada una de las dos formas, es posible encontrar mentalmente la otra realizando algunos pasos meramente por inspección.

Teorema de expansión de Shannon

Los ejemplos previos han demostrado que más de una expresión puede representar la misma función de conmutación. Dos expresiones específicas indicadas son las formas de suma de productos y de producto de sumas. La pregunta que surge es si una función determinada *siempre* puede expresarse en estas formas estándar específicas. Una respuesta, que es el tema de esta sección, la ofreció Claude Shannon.⁹

⁹ Claude E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans AIEE*, 57, 1938, pp. 713-723.

Forma de suma de productos

Suponga que $f(x_1, x_2, \dots, x_n)$ es cualquier función de conmutación de n variables. Shannon demostró que una forma de expresar esta función es

$$f(x_1, x_2, \dots, x_n) = x_1 f(1, x_2, \dots, x_n) + x_1' f(0, x_2, \dots, x_n) \quad (5)$$

En el lado derecho, la función es la suma de dos términos, uno de ellos es importante cuando x_1 toma el valor de 1 y el otro cuando x_1 toma el valor de 0. El primer término es x_1 veces lo que queda de f cuando x_1 toma el valor de 1; el segundo término es x_1' veces lo que queda de f cuando x_1 toma el valor de 0. La prueba de esta expresión es fácil; se obtiene por inducción perfecta. Esto es, si el resultado es verdadero para todos los valores posibles de la variable x_1 (existen sólo dos valores -1 y 0), debe ser verdadero, punto. (Confirme que esto es cierto para todos los valores de x_1 .)

Con toda seguridad usted observa cómo debe proseguir: se repite el proceso en cada una de las funciones restantes, esta vez utilizando otra variable, digamos x_2 . Se continúa el proceso hasta que todas las variables se agotan. El resultado es simplemente una suma de términos, cada uno de los cuales consiste en el producto de n literales multiplicadas por aquello en lo que la función se convirtió cuando cada variable se sustituye ya sea por 0 o 1. Aunque lo último corresponde simplemente al valor de la función cuando las variables toman colectivamente alguna combinación específica de valores; este valor es 0 o 1. De tal modo, el resultado final es una suma de todos los productos no redundantes posibles de las $2n$ literales (las n variables y sus complementos), alguna de las cuales se multiplican por 1 y las restantes por 0. Estas últimas están ausentes simplemente en el resultado final. De acuerdo con Shannon, esto igualará a la función original. La prueba en cada paso se realiza por medio de la inducción perfecta.

El teorema de expansión de Shannon en el caso general es

$$f = a_0 x_1' x_2' \dots x_n' + a_1 x_1' x_2' \dots x_{n-1}' x_n + a_2 x_1' x_2' \dots x_{n-1} x_n' + \dots + a_{2^n-2} x_1 x_2 \dots x_n' + a_{2^n-1} x_1 x_2 \dots x_n \quad (6)$$

Cada a_i es una constante en la cual el subíndice es el equivalente decimal del multiplicador de a_i visto como un número binario. En consecuencia, para tres variables, a_5 (binario 101) es el coeficiente de $x_1 x_2' x_3$.

El teorema de expansión de Shannon en (6) es un resultado fundamental; demuestra que:

Cualquier función de conmutación de n variables puede expresarse como una suma de productos de n literales, una para cada variable.

Ejercicio 8. Sea f una función de dos variables, x_1 y x_2 . Suponga que f toma el valor de 1 en las combinaciones $x_1 x_2$: 00, 10, 11 y el valor de 0 para la combinación restante. Determine la expresión que resulta del teorema de expansión de Shannon aplicado en este caso como complemento. Como tarea adicional, no dependiente del teorema de Shannon, simplifique el resultado si es posible.

Respuesta¹⁰

Al efectuar el proceso de expansión en (6) hasta la terminación, se ha hecho una suposición subconsciente: desde el primer paso hasta el último, después de que alguna (aunque no todas) las variables se han sustituido por constantes, la misma función que queda no es una constante. Esto es, lo que queda sigue dependiente de las variables restantes. Si, en vez de eso, esta función remanente se reduce a una constante, el proceso terminará prematuramente y el término correspondiente tendrá un número menor que n de literales. Ya hemos explicado en la sección anterior

¹⁰ $f = x_1' x_2' + x_1 x_2' + x_1 x_2 = x_1 + x_2'$

x	y	NOT		AND	OR	XOR	NAND	NOR	XNOR
		x'	y'	xy	$x + y$	$x'y + xy'$	$x' + y'$	$x'y'$	$xy + x'y'$
0	0	1	1	0	0	0	1	1	1
0	1	1	0	0	1	1	1	0	0
1	0	0	1	0	1	1	1	0	0
1	1	0	0	1	1	0	0	0	1

Figura 7. Tabla de verdad para operaciones básicas.

cómo reponer cualquier literal faltante. Por consiguiente, la generalización resulta válida incluso en un caso de este tipo.

El teorema de Shannon constituye una "demostración de existencia". Esto es, prueba mediante un procedimiento específico que cierto resultado es verdadero. Una vez que se establece la prueba, entonces para cualquier ejemplo particular no es necesario seguir el procedimiento exacto utilizado para demostrar el teorema. Ahora que sabemos que una función siempre puede ponerse en la forma establecida, es posible investigar maneras más sencillas de hacerlo.

Forma de producto de sumas

El teorema de expansión de Shannon en (6) es una forma de suma de productos. Es fácil presuponer que un resultado similar se cumple para la forma de producto de sumas. No se necesita recorrer una demostración extensiva para esta forma. El resultado se obtiene de (5) y (6) por dualidad. Esto es, se intercambian 0 y 1, al igual que las operaciones de suma y producto. Haciendo esto, las contrapartes de (5) y (6) se vuelven

$$f(x_1, x_2, \dots, x_n) = [x_1 f(0, x_2, \dots, x_n)][x_1' + f(1, x_2, \dots, x_n)] \quad (7)$$

$$f = (b_0 + x_1' + x_2' + \dots + x_n')(b_1 + x_1' + \dots + x_{n-1}' + x_n) \dots (b_{n-2} + x_1 + x_2 + \dots + x_{n-1})(b_{n-1} x_1 + x_2 + \dots + x_n) \dots \quad (8)$$

donde las b_i son las constantes 0, 1. Estas ecuaciones son los duales de (5) y (6). La última muestra que

Cualquier función de conmutación de n variables puede expresarse como un producto de sumas de n literales, una para cada variable.

Si la expansión debe terminar en forma prematura, las literales faltantes siempre pueden reponerse mediante el método ilustrado en la última sección. Las constantes no aparecerán explícitamente en la función real después de que se eliminan las redundancias. Si una de las constantes es 1, por ejemplo, el factor entero correspondiente será 1, puesto que $1 + x = 1$, y el factor correspondiente no estará presente.

Por otro lado, si cualquier constante es 0, en vista de que $0 + x = x$, esta constante no aparece aunque se presentará el factor correspondiente.

Ejercicio 9. Utilice el proceso del teorema de Shannon paso por paso para poner la función del ejercicio 8 en una forma de producto de sumas. Si es posible, simplifique el resultado. •

5 OTRAS OPERACIONES DE CONMUTACIÓN

Se señaló antes que la lógica de proposiciones es isomórfica con el álgebra de conmutación. Como ya se observó, el lenguaje de la lógica de proposiciones ha permeado la manera en la cual se explican las operaciones y las ideas en el álgebra de conmutadores. Tomaremos ahora algunas

otras operaciones que surgen naturalmente en la lógica de proposiciones y las convertiremos para nuestro provecho.

OR exclusiva

En la explicación de la operación OR en la sección 2, se ofreció el ejemplo de una proposición compuesta que parecía ser una proposición OR pero que no lo era: "Ese animal es un gato o un perro." Aunque el animal podría ser un gato como un perro, no es posible que sea *tanto* un gato y un perro. La operación $x + y$ es OR "inclusiva"; ésta incluye el caso en el que tanto x como y son 1. La "o" en "gato o perro" es diferente; tiene un significado diferente en la lógica. Se conoce como OR exclusiva, XOR en forma abreviada, y se le asigna el símbolo \oplus . Así, $x \oplus y$ es verdadera cuando x e y tienen valores de verdad opuestos, pero es falsa cuando x e y tienen el mismo valor. Los valores de verdad de $x \oplus y$ se muestran en la figura 7. (La tabla incluye los valores de verdad de todas las operaciones lógicas básicas, incluso algunas que se presentarán más adelante.)

La OR exclusiva es una función de dos variables. La forma más general de suma de productos para tal función puede escribirse de la manera siguiente:

$$x \oplus y = a_0x'y' + a_1x'y + a_2xy' + a_3xy \quad (9)$$

Los valores de los coeficientes a_i pueden leerse de la tabla de verdad: 0 para a_0 y a_3 , 1 para a_1 y a_2 . De tal modo, (9) se reduce a

$$x \oplus y = x'y + xy' \quad (10)$$

Ejercicio 10. Inicie con la forma de suma de productos para la XOR en (10) para obtener una forma canónica de producto de sumas. ♦

Respuesta¹¹

Operaciones NAND, NOR y XNOR

Además de la operación NOT, tenemos ahora tres en nuestro repertorio: AND, OR y XOR. Es posible crear tres operaciones adicionales negando (complementando o efectuando la operación NOT) de estas tres:

$$\text{NAND (NOT AND):} \quad (xy)' = x' + y' \quad (11)$$

$$\text{NOR (NOT OR):} \quad (x + y)' = x'y' \quad (12)$$

$$\text{XNOR (NOT XOR):} \quad (x \oplus y)' = (x'y + xy')' = xy + x'y' \quad (13)$$

El lado derecho en (13) puede obtenerse de (10) negando los valores de verdad de XOR para obtener los valores de a_i . (Compruébelo.) Los valores de verdad para estas tres se obtienen con facilidad a partir de las tres operaciones de las cuales ellas son el complemento. Los resultados se muestran en la tabla de verdad anterior (figura 7).

Ejercicio 11. Analice la naturaleza de los lados derecho de (11) y (12) como formas de suma de productos o de producto de sumas. ♦

Ejercicio 12. Aplique las leyes de conmutación para determinar una forma de producto de sumas del lado derecho de (13). ♦

¹¹ Sume xx' y yy' , y emplee después la ley distributiva: $(x + y)(x' + y')$. ♦

Advierta en la figura 7 que XNOR es 1 siempre que x e y tienen el mismo valor. En lógica, la función de dos variables que es igual a 1, siempre que las dos variables tienen el mismo valor, recibe el nombre de relación de *equivalencia*. El símbolo que denota a una relación de equivalencia es una doble flecha, $x \Leftrightarrow y$. Así, XNOR y la equivalencia tienen los mismos valores de verdad y se pueden utilizar indistintamente: $A \text{ XNOR } B = A \Leftrightarrow B = xy + x'y'$.

Comparar dos señales (entradas), para ver si son las mismas, constituye una operación importante en muchos sistemas digitales. Así, una compuerta XNOR es un *comparador* de un bit; cuando su salida es 1, sabemos que las dos entradas son iguales. Explicaremos cómo utilizar las compuertas XNOR para comparar números de más de 1 bit en el capítulo 3.

Alguno de los lectores quizá tenga cierta inquietud con la introducción de XOR y las otras operaciones en esta sección. Anteriormente se definió una expresión de conmutación como aquella que incluye los operadores AND, OR y NOT. ¿Quiere decir esto que algo que incluya a XOR o a las otras operaciones presentadas en esta sección no puede ser una expresión de conmutación? Esta contradicción aparente se supera advirtiendo las ecuaciones de la (10) a la (13). Cada una de las operaciones definidas mediante estos operadores se expresa en términos de AND, OR y NOT, por lo que la inquietud debe desaparecer.

6 CONJUNTOS DE OPERACIONES UNIVERSALES

El álgebra de conmutación se presentó en términos de dos operaciones binarias (AND y OR), y una operación unaria (NOT). Toda expresión de conmutación está integrada por variables conectadas mediante diversas combinaciones de estos tres operadores. Esta observación conduce al siguiente concepto:

Un conjunto de operaciones se denomina universal si toda función de conmutación puede expresarse exclusivamente en términos de operaciones de este conjunto.

(Algunos utilizan el término *funcionalmente completo* en lugar de *universal*.) Se concluye que el conjunto de operaciones {AND, OR, NOT} es universal.

En la sección precedente se presentaron varias operaciones más. Dos de éstas son NAND y NOR:

$$x \text{ NAND } y: (xy)' = x' + y' \quad (14)$$

$$x \text{ NOR } y: (x + y)' = x'y' \quad (15)$$

Vemos que cada lado derecho de estas expresiones se indican en términos de sólo dos de las tres operaciones universales (OR y NOT para la primera, y AND y NOT para la segunda). Esto plantea una pregunta general: ¿es posible eliminar una de las tres operaciones del conjunto universal sin que éste deje de serlo? La respuesta es "sí, realmente", si una de las operaciones puede expresarse en términos de las otras dos.

Considere la operación AND, xy . Por medio de la ley de De Morgan, es posible escribirla como

$$xy = (x' + y')' \quad (16)$$

Las únicas operaciones a la derecha son OR y NOT. Puesto que toda AND puede expresarse en términos de OR y NOT, el conjunto {AND, OR, NOT} se expresa en términos del conjunto {OR, NOT}. Conclusión: el conjunto {OR, NOT} es uno universal.

Ejercicio 13. Con el razonamiento anterior, demuestre que el conjunto {AND, NOT} es un conjunto universal.

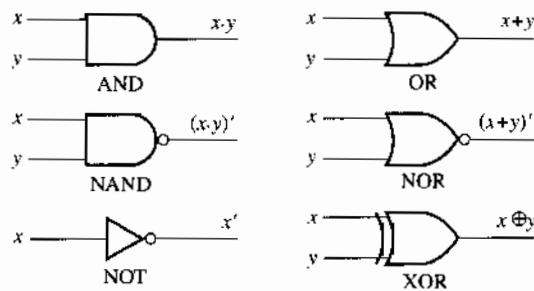


Figura 8. Símbolos para compuertas lógicas.

La conclusión a la que acabamos de llegar es que cualquier función de conmutación puede expresarse en términos de sólo dos operaciones de conmutación. ¿Pero por qué detenernos ahí? ¿Por qué no intentar una más? Exploraremos la operación NAND. Puesto que el conjunto {AND, NOT} es universal, si es posible expresar ambas de estas operaciones en términos de NAND, entonces, ¡{NAND} será universal! Veamos:

$$x' = x' + x' = (xx)' \quad \text{Teoremas 3a y 8a} \quad (17)$$

$$xy = ((xy)')' = [(xy)'(xy)']' \quad \text{Teoremas 2 y 3b} \quad (18)$$

Los lados derechos se expresan en términos únicamente de NAND. Conclusión:

Cualquier función de conmutación puede expresarse exclusivamente en términos de operaciones NAND.

Ejercicio 14. Demuestre mediante un procedimiento similar al que acaba de utilizarse que {NOR} es un conjunto universal. ♦

La conclusión que resulta del ejercicio anterior es:

Cualquier función de conmutación puede expresarse exclusivamente en términos de operaciones NOR.

El punto práctico de la discusión precedente es que, en el mundo real, las operaciones de conmutación se llevan a cabo por medio de dispositivos físicos. Si es posible expresar todas las funciones de conmutación en términos de una sola operación, entonces la implementación física de cualquier función de conmutación puede efectuarse con sólo un tipo de dispositivo físico. Lo anterior tiene implicaciones obvias en cuanto a la simplicidad, conveniencia y costo, las cuales se ampliarán en el capítulo 3.

7 COMPUERTAS LÓGICAS

Hasta ahora hemos trabajado con asuntos bastante abstractos. Hemos explicado las operaciones lógicas en una álgebra denominada de conmutación. Las variables que se manejan en el álgebra de conmutación son variables de conmutación abstractas. Hasta ahora nada se ha dicho que relacione estas variables de conmutación con algo en el mundo real. Además, no se ha establecido conexión entre las operaciones del álgebra de conmutación y los dispositivos físicos.

Estamos listos para cambiar todo lo anterior. Para que el álgebra de conmutación efectúe tareas reales, deben existir dispositivos físicos que realicen las operaciones del álgebra de conmutación con exactitud y con el menor retraso posible. Las variables de conmutación, las cuales toman los valores lógicos de 0 y 1, deben identificarse con señales reales caracterizadas por una variable física, tal como el voltaje.

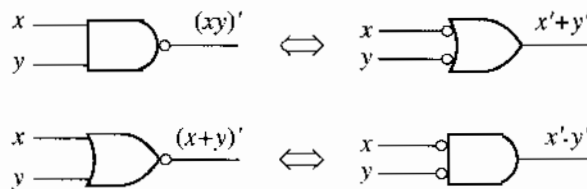


Figura 9. Dos formas equivalentes de compuertas NAND y NOR.

Compuerta es el nombre genérico dado a un dispositivo físico que efectúa cualesquiera de las operaciones de conmutación es *compuerta*. Sin embargo, antes de explicar tales dispositivos, introduciremos otra representación abstracta de las operaciones de conmutación, una representación esquemática para los dispositivos de la vida real y para su interconexión en los circuitos de conmutación reales.

Suponga que cada operación en el álgebra de conmutación se representa mediante un símbolo esquemático diferente, con una terminal distinta para cada variable lógica. En ese caso cualquier expresión de conmutación puede representarse mediante la interconexión de estos símbolos. En el pasado se han realizado intentos para adoptar un conjunto de símbolos estándar. Un conjunto tiene una forma uniforme para todas las operaciones (un rectángulo), con la operación identificada en el interior.¹² Un conjunto de símbolos más común para compuertas utiliza una forma distinta para cada operación, como se muestra en la figura 8.

La compuerta NOT recibe el nombre de *inversor*; cada una de las demás se denomina mediante el nombre de la operación que efectúa: una compuerta AND, una compuerta NOR, etc. Aunque cada compuerta (excepto el inversor) se muestra con dos terminales de entrada, es posible que haya más de dos entradas. (Para los dispositivos reales de los cuales estas compuertas son abstracciones, hay limitaciones prácticas en el número de entradas; estas limitaciones se explicarán un poco más adelante.)

No hay problema al tener más de dos entradas para los dos operadores booleanos básicos AND y OR, debido a que ambos satisfacen la ley asociativa. El resultado es el mismo si sumamos lógicamente x a $(y + z)$ o sumamos $x + y$ primero y luego lo sumamos a z ; sin ambigüedad, el resultado es $x + y + z$. Lo mismo es cierto para la operación XOR. (Compruébelo usted mismo.) Sin embargo, las operaciones NAND y NOR no son asociativas. La salida de una compuerta NAND de tres entradas es $(xyz)'$; sin embargo, esto no es lo mismo que la NAND de x e y seguida por la NAND con z . Esto es, $(xyz)' \neq ((xy)')z'$.

Una compuerta NAND se forma agregando un pequeño círculo (llamado burbuja), representando un inversor, a la línea de salida de una compuerta AND. (Lo mismo es cierto para formar una compuerta NOR a partir de una OR.) Además, poner una burbuja sobre una línea de entrada implica complementar la variable entrante antes de efectuar la operación realizada por la compuerta.

Formas alternativas de las compuertas NAND y NOR

Es posible encontrar otras representaciones de la compuerta NAND y de la compuerta NOR, mediante una aplicación de la ley de De Morgan. Para dos variables, esta ley es:

$$(xy)' = x' + y' \quad (19)$$

$$(x + y)' = x'y' \quad (20)$$

Los lados izquierdo de (19) y (20) se representan mediante los símbolos de compuerta NAND y NOR de la figura 8. El lado derecho de (19) es una operación OR sobre las entradas que se in-

¹² El Instituto de Ingenieros Eléctricos y Electrónicos ha publicado estándares de este tipo (vea IEEE Standard 91-1984). Sin embargo, no se han generalizado.



Figura 10. Formas equivalentes de una compuerta AND.

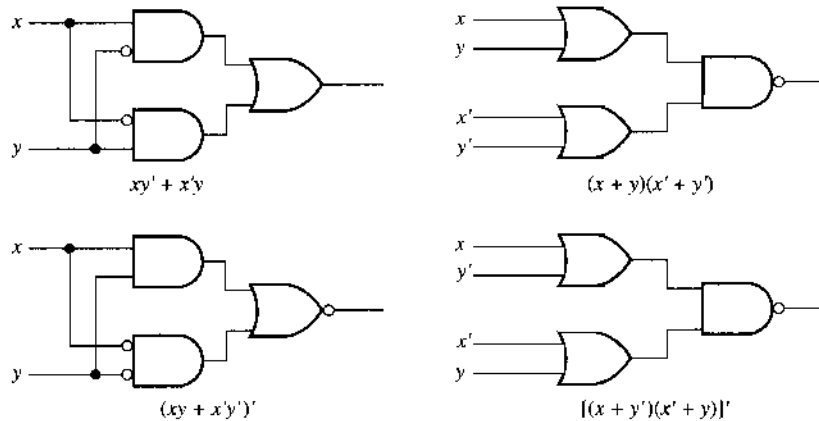


Figura 11. Estructuras alternativas para XOR.

vierte primero. De manera similar, el lado derecho de (20) es una operación AND en entradas invertidas. Estas formas alternativas de compuertas NAND y NOR se muestran en la figura 9.

En cuanto a lo que concierne a compuertas lógicas ideales, las dos formas de compuertas NAND o NOR son igualmente aceptables. Sin embargo, cuando se llega a aplicar la lógica utilizando compuertas físicas reales, existe una diferencia práctica entre las dos; explicaremos esto de manera subsecuente.

Otra cuestión interesante resulta al tomar la inversa de ambos lados en (19):

$$((xy)')' = xy = (x' + y')'$$

Es evidente que una compuerta AND con entradas x e y puede ponerse en práctica con una compuerta NOR cuyas entradas son los complementos de x e y . El resultado se muestra en forma esquemática en la figura 10, donde las burbujas en las entradas de la compuerta NOR representan los inversos de las variables de entrada.

Ejercicio 15. Mediante un procedimiento similar al que se utilizó antes, determine una forma equivalente de una expresión OR y dibuje las compuertas equivalentes correspondientes. ♦

Compuertas OR exclusivas

Es posible escribir varias expresiones de conmutación diferentes para representar la función XOR; algunas de éstas se mostraron anteriormente en este capítulo. Cuatro formas equivalentes son las siguientes:

$$x \oplus y = xy' + x'y \quad (21)$$

$$= (x + y)(x' + y') \quad (22)$$

$$= (xy + x'y')' \quad (23)$$

$$= [(x + y')(x' + y)]' \quad (24)$$

Las dos primeras son las formas canónicas s de p y p de s para la función XOR; éstas se presentaron en (10) en el ejercicio 9. Confirme las otras dos formas.

Coment

8 LÓO

Sin tomar en cuenta la complementación, cada una de las expresiones en el lado derecho incluye tres operaciones de conmutación. En consecuencia, cada una puede representarse mediante tres compuertas interconectadas, como se muestra en la figura 11. Dos de éstas implican la AND o NAND de dos OR, y dos implican la OR o NOR de dos AND. El número de compuertas y literales es el mismo en cada caso.

Comentario

Es posible plantear algunas observaciones con fundamento en lo que acaba de explicarse. Las *compuertas lógicas* constituyen representaciones esquemáticas de las operaciones de conmutación. Una interconexión de compuertas lógicas es exactamente otra forma de escribir una expresión de conmutación. Podemos afirmar que el álgebra de conmutación y las interconexiones de compuertas lógicas son sistemas isomórficos. Cuando usamos los símbolos esquemáticos denominados compuertas lógicas o dibujamos interconexiones de compuertas para representar operaciones de conmutación, no se requiere ninguna implicación de algo físico o real.

Suponga, sin embargo, que los dispositivos físicos que efectúan las operaciones simbolizadas por las compuertas podrían construirse, incluso si no fueran perfectos. En ese caso sería posible construir un circuito físico (al que llamaríamos *circuito de conmutación*) que puede *implementar*, o *realizar*, una expresión de conmutación específica.¹³ El circuito sería una *implementación* o *realización*, de la expresión. Esto es lo que hace al álgebra de conmutadores una estructura matemática abstracta, tan útil en el diseño físico de circuitos reales para efectuar tareas digitales.

Como una simple ilustración, considere una compuerta XOR con entradas x e y . Nos podría interesar comparar las entradas. Resulta claro de (21) que $x \oplus y = 1$ siempre que los valores de las dos entradas sean diferentes. (Lo opuesto sería cierto para una compuerta XNOR.) De tal modo, cualquier compuerta tiene la posibilidad de desempeñar el papel de un comparador de números de 1 bit.

Como otro ejemplo, quizá no sea necesario utilizar de manera explícita un inversor para obtener el complemento de una variable de conmutación. Algunos dispositivos que generan una variable de conmutación también generan su complemento, de manera que ambas están disponibles.¹⁴ En consecuencia, si tanto x como x' aparecen en una expresión, entonces en el circuito correspondiente es posible mostrar líneas de entrada separadas para ambas o indicar el complemento con una burbuja. Si revisa de nuevo la figura 11, observará que cada uno de estos métodos se usa en la mitad de los casos.

8 LÓGICA POSITIVA, NEGATIVA Y COMBINADA

Sin importar cómo se fabrican las compuertas y los circuitos electrónicos, los valores lógicos de 0 y 1 se alcanzan en el mundo real con valores de variables físicas, en la mayoría de los casos voltaje, aunque algunas veces corriente. Los valores particulares de voltaje dependen de la tecnología específica; pero cualesquiera que sean los dos valores, uno es más alto que el otro. Por tanto, uno de ellos se denomina Alto (A) y el otro Bajo (B). No existe una correlación necesaria entre los niveles de voltajes alto y bajo y los valores lógicos de 0 y 1. Son posibles dos esquemas, como se ilustra en las tablas en la figura 12.

Aunque los valores lógicos 0 y 1 no son valores numéricos, si los interpretamos como tales, la lógica positiva (con 1 correspondiente a A) aparece como el esquema natural. La lógica negati-

¹³ Véase el pie de página 4 relativo al significado de *realizar* e *implementar*.

¹⁴ Como se explicará en el capítulo 5, un dispositivo llamado *flip-flop* tiene dos salidas, una de las cuales es el complemento de la otra.

Valor lógico	Nivel de voltaje
0	L
1	H

a)

Valor lógico	Nivel de voltaje
0	H
1	L

b)

Figura 12. Correlaciones entre valores lógicos y niveles de voltaje. a) Lógica positiva. b) Lógica negativa.

tiva es superflua; no aporta nada de valor que no se consiga con la lógica positiva. Por consiguiente, no hay razón útil para adoptarla. Sin embargo, quizá sea útil adoptar la lógica positiva en algunas terminales de dispositivos en un circuito y la negativa en otras terminales. Esta posibilidad se conoce como *lógica combinada*. Su mayor provecho ocurre al trabajar con circuitos físicos reales. Más adelante explicaremos la terminología común en el uso de la lógica combinada.

Se utilizan dos conceptos en torno a los temas de la lógica combinada. Uno es el de *actividad*. La situación normal, podría suponerse, sería la *inactividad*. El teléfono se encuentra normalmente quieto (inactivo) hasta que timbra (se vuelve activo), y entonces es posible llevar a cabo conversaciones. Las luces normalmente están apagadas (inactivas) hasta que se activan mediante un interruptor.

Cuando un microprocesador está efectuando una operación de "lectura", se encuentra activo. En otro caso no está haciendo nada; está inactivo.¹⁵ Asociar el lógico 1 con la actividad y el 0 lógico con la inactividad constituye sólo un hábito.

En las distintas terminales de un sistema físico que se diseñó para efectuar operaciones digitales, los voltajes pueden tener valores ya sea alto (A) o bajo (B). Como acaba de explicarse, las dos maneras en las cuales los niveles de voltaje y los valores lógicos pueden asociarse son $1 \leftrightarrow A$ y $0 \leftrightarrow B$, o lo opuesto $1 \leftrightarrow B$ y $0 \leftrightarrow A$. Lo que complica un poco es que en algunos puntos en un circuito es posible hacer la asociación de una manera y en otros puntos de la manera opuesta.

Debido a la asociación de la actividad, si se hace la asociación de $1 \leftrightarrow A$, el esquema consiste en afirmar que se está en "activo en nivel alto". Si se realiza la asociación $1 \leftrightarrow B$, se dice que se estará en "activo en nivel bajo". No obstante, esta descripción emplea dos pasos, dos niveles de asociación: la actividad se conecta primero con el 1 lógico; luego el 1 lógico, a su vez, se asocia con un voltaje alto o uno bajo. De manera esquemática, la secuencia de pensamiento es:

actividad \rightarrow 1 lógico \leftrightarrow alto voltaje = activo en nivel alto

actividad \rightarrow 1 lógico \leftrightarrow bajo voltaje = activo en nivel bajo

"Activo en nivel alto" significa que el 1 lógico está asociada con el alto voltaje. De manera más simple, podría decirse 1-alto o 1-A en lugar de "activo en nivel alto". El adjetivo *activo* sólo añade distancia a la asociación real que se va a establecer entre un valor lógico y un nivel de voltaje.

Otro concepto utilizado algunas veces como un sinónimo de *actividad* es la noción de *aseveración*. Así, "aseverar en alto" y "aseverar en bajo" significa lo mismo que "activo en nivel alto" y "activo en nivel bajo" que, al final, significan la asociación, en la lógica positiva, con un alto voltaje y con un bajo voltaje, respectivamente. El razonamiento se realiza de manera similar a lo siguiente: aseverar es afirmar, para establecer que algo es así; lo que se asevera podría pensarse como "verdadero". En la lógica de proposiciones, una proposición es verdadera o falsa. De manera que cuando algo se asevera, esto es equivalente a decir que la proposición es ver-

¹⁵ Incluso podría pensarse en la primera ley de Newton para ejemplificar este argumento. Normalmente, las cosas están estáticas; si desea cambiar algo, tendrá que ponerse en acción y ejercer una fuerza.

dadera. La dicotomía verdadero/falso en la lógica de proposiciones se asocia con las constantes de conmutación 1 y 0. Pero no hay una correspondencia uno a uno entre 1-V y 0-F; simplemente es costumbre asociar 1 con "verdadero" y 0 con "falso". Asociar con 1:

$$\text{aseverado} \Rightarrow \text{verdadero} \Rightarrow 1 \text{ lógico}$$

Por consiguiente, decir que algo se "asevera en alto" significa que 1 corresponde a un alto voltaje. El uso de "aseverado" no agrega nada aquí a la identificación de 1 con un alto voltaje. Podría decirse también "1-alto" o 1-A. De la misma manera, señalar que algo es "aseverado en bajo" significa que 1 corresponde a un bajo voltaje. Aquí el término "aseverado" no añade nada a la identificación de 1 con un bajo voltaje; se podría señalar también 1-B.

Un comentario final sobre la terminología. Usualmente el verbo "aseverar" no se asocia con "alto" o "bajo". Podría decirse, por ejemplo, que cierta señal debe aseverarse antes de que otra cosa pueda suceder. En tal uso, no hay compromiso para aseverar en alto o aseverar en bajo, sólo para aseverar. Esta terminología no requiere un compromiso con la lógica combinada o la lógica positiva; es posible aplicarla en cualquier caso. Por consiguiente, puede ser una terminología útil.

Cuando los dispositivos físicos se usan para poner en práctica un diagrama lógico, es posible utilizar una correspondencia distinta de niveles de voltaje con valores lógicos en diferentes terminales de entrada y salida, incluso en puntos internos. En la fase de implementación real de un diseño, tal vez resulte útil considerar lo que se ha llamado lógica combinada, en la cual se hace la correspondencia 1-A en algunas terminales del dispositivo y la 1-B en otros. Para lograr este propósito, se usan convenciones y notaciones especiales para transmitir la información a aquellas terminales que corresponde a 1-A y a aquellas que corresponden a 1-B.

En este libro, nos adheriremos a la lógica positiva, de modo que no trabajaremos con esta notación especial. En realidad, los símbolos de compuerta que se muestran en las figuras 8-11 se basan en la lógica positiva.

9 ALGUNAS CUESTIONES PRÁCTICAS RELATIVAS A COMPUERTAS¹⁶

Hasta este momento hemos adoptado el siguiente punto de vista: las especificaciones de una tarea lógica, seguidas por los procedimientos del álgebra de conmutación, producen una expresión de conmutación. Un diagrama esquemático que contiene compuertas lógicas se construye entonces para realizar esta expresión. Al final, esperamos construir la encarnación física de este diagrama utilizando dispositivos físicos reales, generalmente conocidos como *hardware*.

La manera en la cual las compuertas físicas se diseñan y construyen depende de la tecnología de ese momento. Como ya se mencionó, los primeros circuitos de conmutación utilizaban dispositivos mecánicos: interruptores y relevadores.¹⁷ Los primeros dispositivos de conmutación que recibieron el nombre de "compuertas" se diseñaron con tubos de vacío. Estos últimos se sustituyeron después por diodos semiconductores y, posteriormente, por transistores bipolares y luego MOSFET. Cada compuerta se construía individualmente con componentes discretos.

¹⁶ Esta sección es bastante descriptiva. Aunque se intercalan un par de ejercicios en ella, éstos no requieren mucho esfuerzo y creatividad para terminarlos. Lo que se explica es sumamente importante para el trabajo de laboratorio. Si lo desea, puede ser usted un lector muy pasivo; pero lo invitamos a que se comprometa, tome notas, plantee preguntas y consulte otros libros y manuales de fabricantes en cuanto a aspectos específicos.

¹⁷ Si bien los primeros interruptores eran dispositivos mecánicos, los dispositivos electrónicos modernos más fundamentales actúan también como interruptores; a saber, transistores, tanto el tipo de unión bipolar como, en especial, la variedad MOSFET. (Consulte el apéndice para las descripciones de los MOSFET y los BJT.) Por consiguiente, la implementación de circuitos de conmutación con interruptores puede actualizarse utilizando MOSFET. Las operaciones lógicas AND y OR se llevarían a cabo entonces con conexiones en serie y en paralelo de tales interruptores. Puesto que la mayor parte de los proveedores actuales tienen un interés consumado en la tecnología que se usa en el presente, es poco probable que se haga un interruptor. (¡Dios nos libre!) Ya, en serio, algunos, sin embargo, están regresando a los circuitos de interruptores en los cuales Shannon fue pionero, y algunos libros están empezando a reintroducir tales circuitos.

Designación	Nombre	Potencia	Velocidad
LS	Schottky de baja potencia	Baja	Lenta
ALS	LS avanzada	Baja	Moderado
S	Schottky	Media	Rápida
AS	Schottky avanzada	Media	Rápida
L	Baja potencia	Baja	Moderado
F	Rápida	Alta	Muy rápida

Figura 13. Algunas subfamilias dentro de la familia de compuertas TTL.

El advenimiento de los circuitos integrados permitió, primero, que una compuerta completa se fabricara como una unidad, y después como varias compuertas independientes. Tales unidades *integradas a pequeña escala* (SSI) aún se utilizan. Rápidamente fue posible incorporar en el mismo circuito integrado una interconexión de muchas compuertas que en conjunto efectuaban ciertas operaciones específicas.

Estudiaremos varios de tales circuitos *integrados a media escala* (MSI) en el capítulo 4. Con el tiempo, un circuito completo compuesto por cientos de miles de compuertas —como un microprocesador— se llegó a fabricar como una sola unidad, sobre una sola “pastilla”. Algunas características de los circuitos integrados se explicarán aquí. El diseño con circuitos específicos de este tipo se realizará en el capítulo 8.

Familias lógicas

Como se mencionó antes brevemente, el diseño e implementación de las compuertas lógicas en cualquier área determinada se lleva a cabo utilizando los dispositivos particulares disponibles en ese momento.

Cada tipo de dispositivo opera de manera óptima con valores específicos del voltaje del suministro eléctrico. Es posible que diferentes diseños originen diferentes niveles de alto y bajo voltajes. Un conjunto de compuertas lógicas que utiliza una tecnología de diseño único se conoce como *familia lógica*. Algunas que se consideraron “avanzadas” hace apenas cuatro décadas (como la lógica resistor-transistor, o la familia RTL) ahora son obsoletas y se pueden encontrar en los museos.

Diferentes diseños han sido creados y promovidos por distintos fabricantes y resultan adecuados para diferentes requerimientos. La familia ECL (lógica de emisor acoplado) provino de Motorola, en tanto que Texas Instruments es el creador de la familia TTL (lógica de transistor-transistor). Con el tiempo, el diseño TTL básico se ha mejorado y modificado de diferentes maneras para superar una propiedad de una compuerta (digamos la velocidad) a costa de alguna otra propiedad (por ejemplo, el consumo de potencia). En esta forma, se crearon diferentes subfamilias dentro de la familia TTL. Una tabla que lista el número de subfamilias TTL se presenta en la figura 13.

Si bien las familias lógicas TTL y ECL utilizan transistores bipolares como los elementos de conmutación, la tecnología CMOS (semiconductor-óxido-metal complementario) recurre al transistor MOSFET. Las subfamilias de CMOS se listan en la figura 14. Una pregunta importante surge en cuanto a si las entradas y salidas de las compuertas CMOS pueden interconectarse con las compuertas TTL sin ningún tipo de circuitos de conversión especiales.

Si eso es posible, afirmamos que las familias lógicas CMOS son compatibles con TTL. Resulta que con un suministro eléctrico de entre 3.3 y 5 volts, las familias CMOS son en realidad compatibles con TTL.

Los circuitos ECL, por otro lado, no son compatibles ni con TTL ni con CMOS. Se requieren circuitos de conversión especiales para conectar las compuertas ECL con las compuertas

Designación	Nombre	Potencia	Velocidad
HC/HCT	CMOS de alta velocidad	Muy baja	Moderada
AC/ACT	CMOS avanzada	Baja	Rápida
C	CMOS	Muy baja	Moderada
LCX/LVX/LVQ	CMOS de bajo voltaje	Muy baja	Moderada
VHC/VHCT	CMOS de muy alta velocidad	Baja	Rápida
CD4000	CMOS	Baja	Moderada

Figura 14. Algunas subfamilias dentro de la familia de compuertas CMOS.

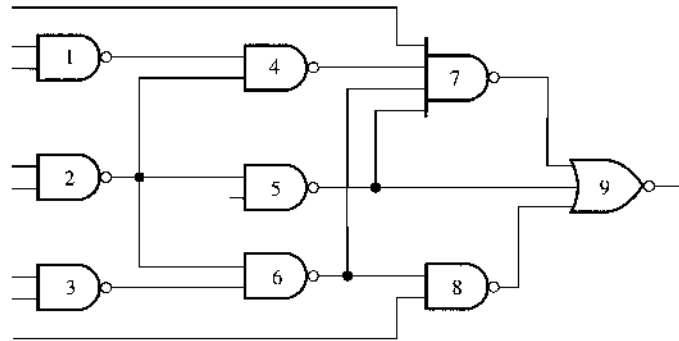


Figura 15. Circuito lógico que ilustra la carga de compuertas.

TTL o CMOS. Salvo por la subfamilia CD4000, las familias CMOS son compatibles en patillas o terminales físicas de conexión con las familias TTL de la figura 13. Esto es, dos patillas con la misma funcionalidad tienen la misma asignación de patillas para las entradas, salidas, potencia y conexión a tierra. ¿Cómo se decide qué subfamilia usar en un caso determinado? Lo que se necesita es una métrica que tome en cuenta tanto la velocidad como el consumo de potencia. Podría considerarse el producto de estos dos valores, aunque eso no funciona puesto que la meta es aumentar uno y reducir el otro. En vez de la velocidad, usamos el retardo de una señal al recorrer una compuerta; un retardo bajo significa alta velocidad. De tal modo, el producto *retardo-potencia* se usa como una métrica. Cuanto menor es este producto, tanto mejor.

En este libro no estudiaremos la operación de los dispositivos electrónicos con los cuales se construyen las compuertas lógicas. Sin embargo, utilizaremos las familias de compuertas TTL y CMOS con fines ilustrativos, aunque sólo de manera descriptiva.

Características de entrada/salida de compuertas lógicas¹⁸

Los circuitos de conmutación físicos se integran a partir de interconexiones de compuertas lógicas físicas de acuerdo con una expresión de conmutación obtenida para lograr una tarea digital particular. Un ejemplo de un circuito se muestra en la figura 15. La salida de una compuerta se convierte en la entrada a una o más compuertas distintas.

Idealmente, no existiría interacción entre compuertas; esto es, la operación de la compuerta 4, por ejemplo, no tendría influencia en la operación propia de la compuerta 2. Sin embargo, las entradas y las salidas de la compuerta se conectan a dispositivos electrónicos internos a las com-

¹⁸ Esta subsección no es esencial para lo que sigue. Incluso si su conocimiento previo no le permite asimilarse del todo, debe leerla de cualquier modo, al menos para familiarizarse con la terminología. Para información adicional consulte el *Data Book for Design Engineers* de Texas Instruments.

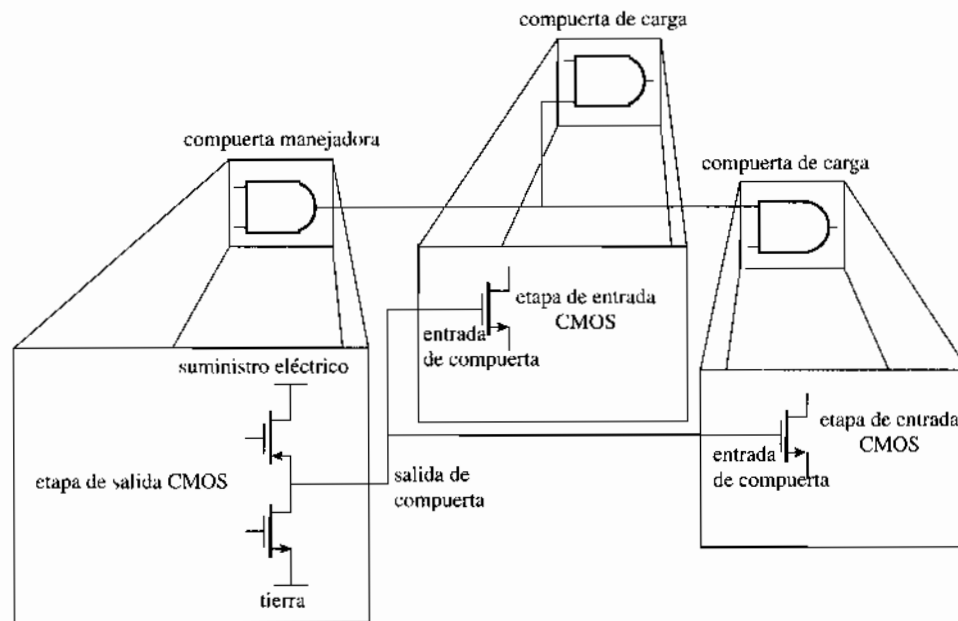


Figura 16. Etapas características de entrada/salida de compuertas lógicas CMOS.

puertas. Cuando estos dispositivos están conduciendo, las corrientes fluyen hacia adentro y hacia fuera de las terminales de la compuerta. Estas corrientes deben necesariamente fluir a través de las terminales de salida de la compuerta precedente o de las terminales de entrada de la compuerta que sigue. Puesto que existen límites prácticos impuestos por las propiedades de los dispositivos electrónicos sobre el nivel de las corrientes que pueden conducirse, existen límites en el grado al cual las compuertas pueden interconectarse. Los diseñadores lógicos no necesitan conocer los detalles de la tecnología de semiconductores y de la implementación de las compuertas lógicas, pero sí comprender las características de entrada/salida de la tecnología en uso para ser capaces de analizar las interacciones entre compuertas.

Las interacciones entre compuertas son específicas de la tecnología, por lo que es necesario distinguir CMOS y TTL. Vamos a analizar primero las características de entrada/salida de CMOS y después de TTL. Una salida de compuerta lógica es llevada a un nivel alto o bajo como una función de los estados de la entrada. De acuerdo con el apéndice sabemos que el MOSFET es un interruptor controlado por voltaje. Así, las entradas a una compuerta lógica CMOS se conectan a las terminales de compuerta de los MOSFET. Cuando una entrada está en estado uno, cierra un interruptor MOSFET; en el estado alternativo, abre un interruptor MOSFET. La salida de una compuerta CMOS contiene uno o más dispositivos MOSFET configurados para llevar la salida a nivel alto para ciertos estados de entrada. Si las entradas son tales que la salida está en alto, entonces se cierra un conjunto de interruptores entre la salida y el suministro eléctrico, conectando la salida a dicho suministro. (El voltaje del suministro eléctrico corresponde por ello a 1 lógico, en lógica positiva.)

De modo similar, una compuerta CMOS contiene uno o más MOSFET para llevar la salida a nivel bajo cuando se requiera. Estos MOSFET conectan la salida a tierra (lo que corresponde a 0 lógico) para ciertas entradas. En nuestro análisis podemos suponer que se usa un MOSFET para llevar la salida a alto y un MOSFET para conducir la salida a bajo. Las conexiones MOSFET características de entrada y salida para la tecnología CMOS se presentan en la figura 16.

Cuando se activa un MOSFET, no se comporta como un corto circuito, sino como una fuente de corriente. Éste es un punto importante. Si se comportara como un corto circuito, ¡las com-

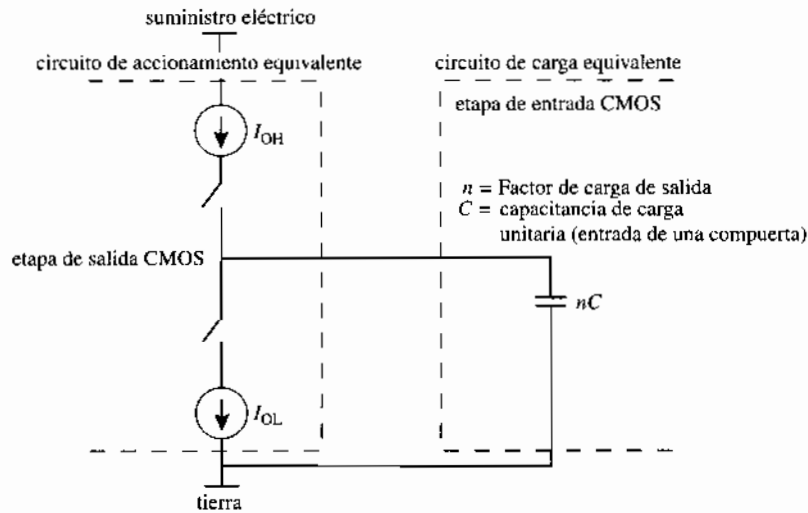


Figura 17. Circuito equivalente para el análisis de conexiones de compuertas lógicas en la tecnología CMOS.

puertas tendrían retardo cero y disipación de potencia cero! Sabemos que el MOSFET tiene muy alta impedancia de entrada entre la compuerta y tierra (véase el apéndice). El aislante de la compuerta forma el dieléctrico de una capacitancia, y es esta misma la que crea el retardo en una compuerta. Una conexión de la salida de una compuerta a las entradas de otras compuertas puede modelarse como se indica en la figura 17.

En una compuerta lógica CMOS los interruptores ilustrados en la figura 17 nunca se cierran al mismo tiempo. Por tanto, una de las corrientes fluye hasta que el voltaje de salida (que es proporcional a la carga en el capacitor) alcanza su valor final. Cuando el circuito se encuentra en el estado estable, no circula ninguna corriente. Cuando el circuito cambia de estado, la capacitancia se carga o descarga al nuevo estado. La diferencia entre las interacciones de entrada/salida de la CMOS y de TTL se debe principalmente a la diferencia en las etapas de entrada de las dos compuertas lógicas. Las etapas de entrada/salida de TTL se muestran en la figura 18.

Una entrada de una compuerta lógica TTL se conecta al emisor de un transistor bipolar, de manera que la carga vista por una salida de compuerta es resistiva (opuesto a la capacitiva para CMOS). Los transistores bipolares en la etapa de salida de una compuerta TTL se comportan como una fuente de corriente cuando se activan, y como un circuito abierto al momento de desactivarse. El circuito equivalente para el análisis de la interacción de compuertas en TTL se muestra en la figura 19.

Factor de carga de salida y Factor de carga de entrada

Podemos aprender algunas cuestiones haciendo ciertas observaciones acerca de la estructura de la figura 15. La salida de la compuerta 1 va exclusivamente a la entrada de la compuerta 4. Decimos que la compuerta 1 *acciona* la compuerta 4 e, inversamente, que la compuerta 4 *carga* la compuerta 1. La salida de la compuerta 2 va a las entradas de las otras tres compuertas: la compuerta 2 *acciona* las compuertas 4, 5 y 6, y cada una de las últimas compuertas carga a la compuerta 2. El número de entradas de compuerta accionadas, o cargadas, por la salida de una sola compuerta se conoce como *factor de carga de salida* de la compuerta accionadora. El factor de carga de salida de la compuerta 2 en la figura 15, por ejemplo, es 3.

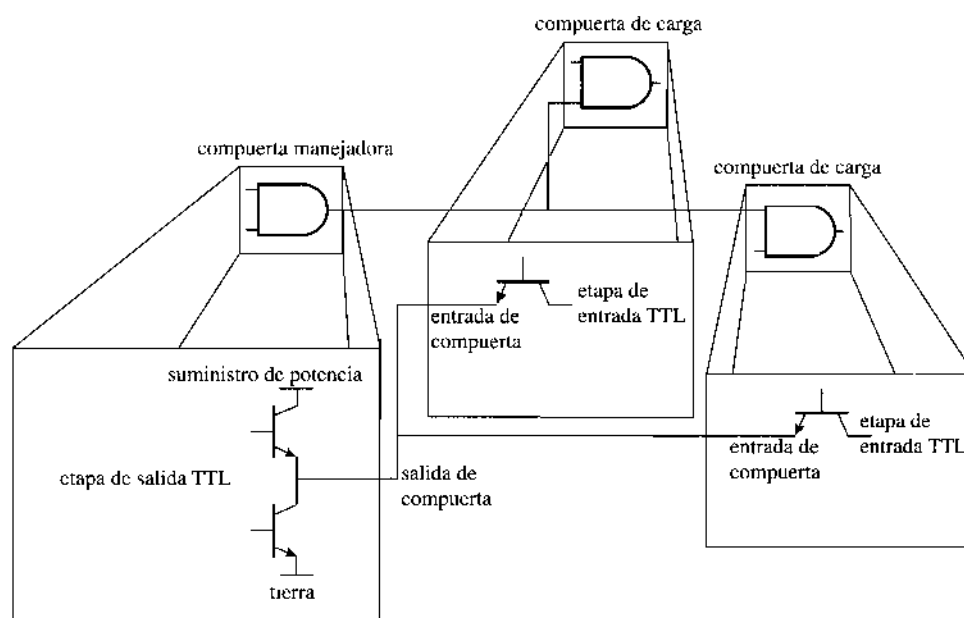


Figura 18. Etapas características de entrada/salida de compuerta lógicas TTL.

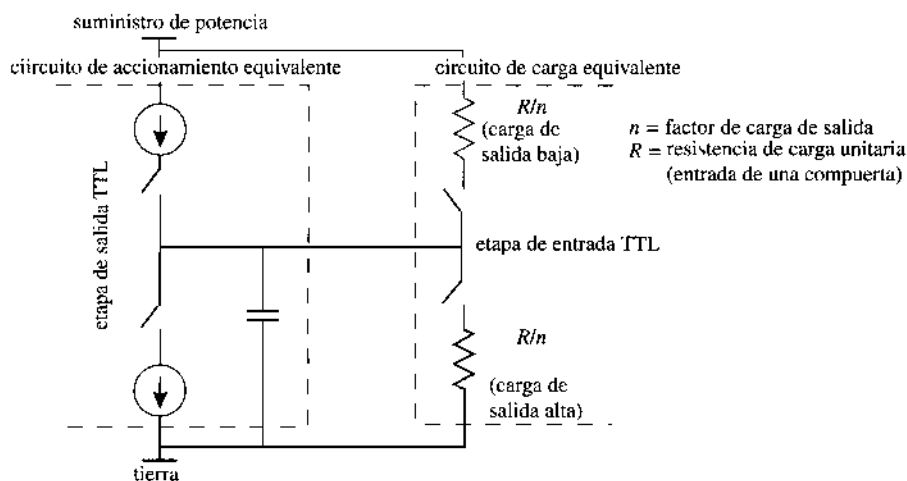


Figura 19. Circuito equivalente para analizar conexiones de compuerta lógicas TTL.

En la tecnología CMOS los transistores de una salida de compuerta no tienen que suministrar ninguna corriente estática a las entradas de las compuertas de carga. Así, una compuerta CMOS puede tener un factor de carga de salida ilimitado sin afectar la lógica (nivel de voltaje) del circuito. Sin embargo, como se explicará más adelante, el factor de carga de salida en la tecnología CMOS tiene un impacto sustancial en el retardo del circuito.

En circuitos TTL, una salida de compuerta debe suministrar una corriente estática a las compuertas que está manejando. Por ejemplo, en la figura 15 cuando la salida de la compuerta 2 corresponde al nivel bajo, según el último párrafo de la subsección precedente, están conduciendo los transistores en las entradas de las compuertas manejadas (4, 5 y 6). Todas sus corrientes deben fluir por el transistor de salida de la compuerta accionadora (2). Este proceso de regresar la corriente en la entrada de una compuerta hasta tierra a través del transistor de salida de la com-

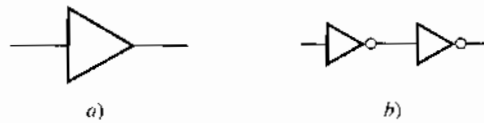


Figura 20. Búfer.

puerta accionadora recibe el nombre de *drenado* de la corriente. Para una familia determinada de compuertas TTL, cuando el transistor de entrada está conduciendo, la cantidad de corriente se denomina *carga estándar*. La compuerta 2 en la figura 15 debe drenar tres cargas estándar puesto que sus salidas manejan las entradas de otras tres compuertas. La operación adecuada del transistor de salida de la compuerta 2 impone un límite en el número de cargas estándar que puede drenar. Así, en cada diseño de circuito TTL hay un factor de carga de salida máximo, que no debe excederse.

Al continuar la inspección de la figura 15, advertimos que la mayor parte de las compuertas tienen dos entradas, aunque la compuerta 9 tiene tres entradas y la compuerta 7 tiene cuatro entradas. Nos referimos al número de entradas a una compuerta como su *factor de carga de entrada*. Si bien no hay una fuerte limitación en el factor de carga de entrada de una compuerta impuesta por su propia operación —como la hay en el factor de carga de salida— existe, sin embargo, una consideración práctica. Las compuertas se fabrican con factor de carga de entrada específicos, un número fijo de terminales de entrada. Si el diseño lógico de un circuito demanda una compuerta que no se dispone comercialmente con un factor de carga de entrada alto, el diseño debe modificarse para utilizar compuertas con factor de carga de entrada disponibles. Se presentan diferencias entre las restricciones de factor de carga de entrada de las familias CMOS y TTL, pero están más allá del objetivo de este texto. La restricción del factor de carga de entrada en la CMOS se vuelve a considerar en la subsección sobre velocidad y retardo de la propagación.

Búfers o Reforzadores

En circuitos TTL cuando el factor de carga de salida de una compuerta es alto, la compuerta necesitará drenar gran cantidad de corriente. Lo anterior puede ser la causa de que la compuerta se sobrecargue y deje de funcionar apropiadamente. En los circuitos CMOS los grandes factores de carga de salida pueden provocar un incremento en el retardo. Para superar estas dificultades, se introduce un *búfer* (no inversor) a la salida de la compuerta; su función es simplemente proporcionar una corriente de accionamiento incrementada que permita la operación continua con una carga mayor. El búfer no efectúa operaciones lógicas, excepto la trivial operación identidad. Esto es, su salida es la misma que su entrada. El símbolo lógico para el búfer se muestra en la figura 20a; corresponde a un inversor sin la burbuja.

Podría constar de dos inversores en cascada, como se indica en la figura 20b. De esta manera, tanto una variable como su complemento están disponibles para accionar una carga sustancial.

Consumo de potencia

Siempre que se presenta una corriente en un circuito físico, parte de la potencia eléctrica asociada se convertirá en calor. Si el circuito siguiera funcionando, dicho calor debe disiparse de manera que no se produzca un calentamiento excesivo de los dispositivos en el circuito. El diseño de compuertas y circuitos integrados —incluso los requerimientos del suministro de potencia, el número y el tipo de transistores, así como otros factores— tendrán mucho que ver con el consumo de potencia. Es posible diseñar circuitos específicamente para bajo consumo de potencia. El lector debe saber actualmente, sin embargo, que no puede obtener algo a cambio de nada. El bajo consumo de potencia se consigue a un precio, casi siempre a expensas de una reducción de velocidad. Repase las tablas de la familia TTL en la figura 13 y de la familia CMOS en la figura 14; advierta el intervalo de disipación de potencia con base en la diversidad de diseños.

Margen de ruido

Cada familia lógica tiene voltajes nominales específicos que corresponden a su nivel alto o a su nivel bajo. (En el caso de la familia TTL, el valor nominal alto es de salida 2.0 V y el valor nominal de salida bajo corresponde a 0.4 V.) Sin embargo, los circuitos digitales operan confiablemente aun cuando los niveles de voltaje se desvíen en forma considerable de sus valores nominales. Además, el nivel de salida alto de una compuerta no necesariamente es el mismo que su nivel de entrada alto. Lo que se considera un voltaje de entrada o salida "alto" se extiende en una gama de valores y lo mismo ocurre para los valores "bajos".

Las entradas y las salidas de las compuertas que se están analizando son señales con las que pretendemos contar. No obstante, existen muchas formas en las cuales es posible que se generen señales indeseables en un circuito y que por ello cambien los valores de entrada y salida respecto a los que se proponen. Denominaremos *ruido* a estas señales extrañas que se producen en un sistema. El ruido se genera por medio de una variedad de mecanismos dentro del ambiente en el cual opera el circuito, desde la radiación atmosférica hasta la interferencia del sistema de suministro eléctrico de 60 Hz. Es posible que el ruido se genere también dentro de los dispositivos electrónicos en el propio sistema.

Cuando las señales propuestas están acompañadas por ruido, las señales reales serán versiones modificadas de las deseadas. Deben tomarse providencias en el diseño del circuito para permitirle que continúe operando en la presencia de ruido hasta cierto nivel previsto. Esto es, debe haber cierta inmunidad al ruido. Una parte de la cantidad de ruido que se tolerará antes del mal funcionamiento del circuito es el *margen de ruido* (N). No seguiremos con los detalles de este tema. Basta decir que lo que estamos considerando niveles de alto voltaje y niveles de bajo voltaje en la operación de compuertas no son valores fijos sino intervalos de valores. Mientras los voltajes de entrada y salida se mantengan dentro del intervalo de valores especificados por el fabricante, aun cuando se contaminen por ruido, las compuertas operarán como si los voltajes tuvieran sus valores nominales.

Velocidad y retardo de propagación

La velocidad a la cual operan los circuitos lógicos determina qué tan rápido completan éstos una tarea. Las limitaciones en la velocidad provienen de dos fuentes:

- El retardo que se encuentra por una señal que atraviesa una sola compuerta.
- El número de *niveles* en el circuito, esto es, el número de compuertas que una señal encuentra al ir de una entrada a la salida. Denominamos *trayectoria lógica* a las secuencias de compuertas desde una entrada hasta una salida en un circuito.

Ejercicio 16. Reexamine la figura 15. Especifique el número de niveles que cada entrada encuentra en su camino hacia la salida.

Respuesta¹⁹

El retardo en una compuerta TTL proviene en gran medida del hecho de que los transistores dentro de la misma requieren un tiempo distinto de cero para conmutar de un estado indeterminado a una condición conductora y viceversa. Este retardo, en su mayor parte, es independiente de la carga vista por una compuerta. De tal modo, en circuitos TTL puede suponerse que una compuerta tiene un retardo fijo, y el retardo total de una entrada lógica a una salida lógica puede estimarse acumulando los retardos de las compuertas en la trayectoria lógica desde la entrada hasta la salida.

¹⁹ Seis señales encuentran el máximo de cuatro niveles, una señal encuentra tres niveles y dos señales encuentran un nivel.

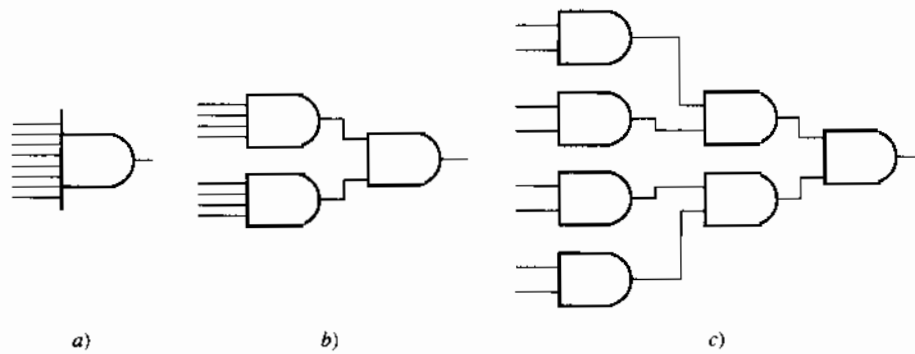


Figura 21. Tres realizaciones alternativas de una compuerta AND de ocho entradas.

El retardo en una compuerta CMOS no resulta sólo del tiempo requerido para que los transistores conmuten entre los estados conductor y no conductor, sino también del tiempo que se requiere para cargar y descargar la capacitancia de carga de las entradas de las compuertas que maneja. Vamos a llamar al retraso ocasionado por la conmutación de estado de los transistores en una compuerta, el *retardo intrínseco* de la compuerta, y al debido a la capacitancia de carga, *retardo extrínseco*.

El retardo intrínseco depende fuertemente del factor de carga de entrada de una compuerta; el retardo extrínseco depende fuertemente de su factor de carga de salida. Las compuertas con un factor de carga de entrada grande tienen un retardo intrínseco mayor que las compuertas con factor de carga de entrada pequeño.

En CMOS quizá resulte ventajoso aumentar el número de niveles de compuerta para mantener baja el abanico de entrada a fin de reducir el retardo de un circuito. Por ejemplo, en la figura 21 se presentan tres realizaciones de una compuerta AND de ocho entradas. En la tecnología CMOS (debido a las características del MOSFET) es probable que las realizaciones ilustradas en las figuras 21b y 21c tengan retardos inferiores que los correspondientes a la figura 21a; éstos, sin embargo, dependen de las características de la subfamilia específica de CMOS utilizada para la implementación. El retardo extrínseco lo provocan las restricciones físicas impuestas por la capacitancia: la corriente de la compuerta accionadora requiere un tiempo para cargar o descargar la capacitancia de carga hasta el nivel de voltaje deseado. No es posible estimar con exactitud el retardo en una compuerta CMOS contando simplemente el número de niveles de compuertas en una trayectoria lógica. Este retardo para una capacitancia de carga determinada se obtiene de las hojas de datos de los fabricantes, las cuales contienen curvas de retardos medidos como una función de la capacitancia de carga.

La respuesta transitoria que se muestra en la figura 22 ilustra el efecto integrado de todos los transistores y de los demás componentes de una compuerta. Los símbolos t_{pBA} y t_{pAB} se usan casi

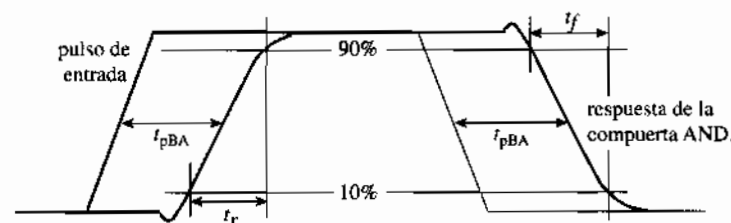


Figura 22. Formas de onda de la salida de una compuerta AND de dos entradas en respuesta a un pulso en una entrada, con la otra entrada mantenida en 1 lógico.

siempre para denotar el retardo de propagación de bajo a alto y de alto a bajo a través de una compuerta, respectivamente. El tiempo de retardo de la propagación es el que transcurre entre la aplicación de una señal de entrada y la respuesta de salida. Los tiempos de retardo t_{pBA} y t_{pAB} no son necesariamente iguales para una compuerta específica. El retardo de la trayectoria lógica se puede determinar mediante la acumulación de t_{pBA} y t_{pAB} para cada compuerta en la trayectoria. Los símbolos t_r y t_f denotan, respectivamente, los tiempos de ascenso y caída de una señal y se definen como el tiempo requerido para que una señal realice una transición de 10 a 90% de su valor final.

10 CIRCUITOS INTEGRADOS

Históricamente, tanto los circuitos analógicos como los digitales, desde el dispositivo más simple hasta el más complejo, se construyeron con componentes discretos interconectados mediante alambres conductores denominados *hilos de conducción*. Durante la década de los años 60 se desarrolló un método para fabricar todos los componentes que conformaban un circuito como una sola unidad sobre una “pastilla” de silicio. El nombre genérico de este proceso es “integración” del circuito. Así, un *circuito integrado* (CI) efectúa alguna función; consta de múltiples componentes eléctricos (lo que incluye transistores, resistores y otros), interconectados propiamente sobre una pastilla semiconductor durante el proceso de fabricación.²⁰

La complejidad de un circuito digital fabricado como un CI varía de unas cuantas compuertas a un microprocesador completo sobre una sola pastilla. Las siguientes clasificaciones dan una medida de la complejidad.

- SSI (siglas en inglés de *integración a pequeña escala*): CI con hasta una docena de compuertas por pastilla, a menudo copias múltiples del mismo tipo de compuerta.
- MSI (siglas en inglés de *integración a media escala*): CI con unos cuantos cientos de compuertas por pastilla, interconectadas para efectuar alguna función específica.
- LSI (siglas en inglés de *integración a gran escala*): CI con algunos cientos de compuertas por pastilla.
- VLSI (siglas en inglés de *integración a muy grande escala*): CI con más de 10,000 compuertas por pastilla.²¹

Cada circuito integrado se empaqueta como una sola unidad con varias terminales (pastillas) disponibles para conectar entradas, salidas y suministros de potencia eléctrica. Unos cuantos circuitos integrados SSI característicos se ilustran en la figura 23. Consulte los manuales de los fabricantes para otros casos.

Algunas características de los CI

Los circuitos digitales que utilizan CI tienen varias características altamente significativas:

Tamaño y espacio. El material de base de un circuito integrado es una *oblea* semiconductor de casi 100 mm de diámetro y de sólo 0.15 mm de espesor. Cada oblea se subdivide en muchos cientos de sectores rectangulares denominadas pastillas o *chips*. Sobre cada pastilla se fabrica un circuito completo. Aunque cada circuito se monta en un empaque de tamaño estándar, se obtiene una considerable ventaja en el tamaño.

²⁰ Si visita un museo histórico, entre las colecciones encontrará objetos como cuchillos de huesos milenarios, puntas de flecha de pedernal y martillos formados por una piedra atada con corteza a una rama de árbol. Con miles de años de antigüedad, estos objetos ahora se conocen como *primitivos*. De la misma manera, las compuertas simples, los búfers o reforzadores y los inversores se describen como *dispositivos primitivos*, aunque éstos se crearon apenas hace unas cuantas décadas. Lo anterior le indica algo acerca de la rapidez del cambio en la época contemporánea.

²¹ Algunos en Inglaterra sugieren una categoría adicional: VLSII, ¡lo que representa en verdad a VLSI!

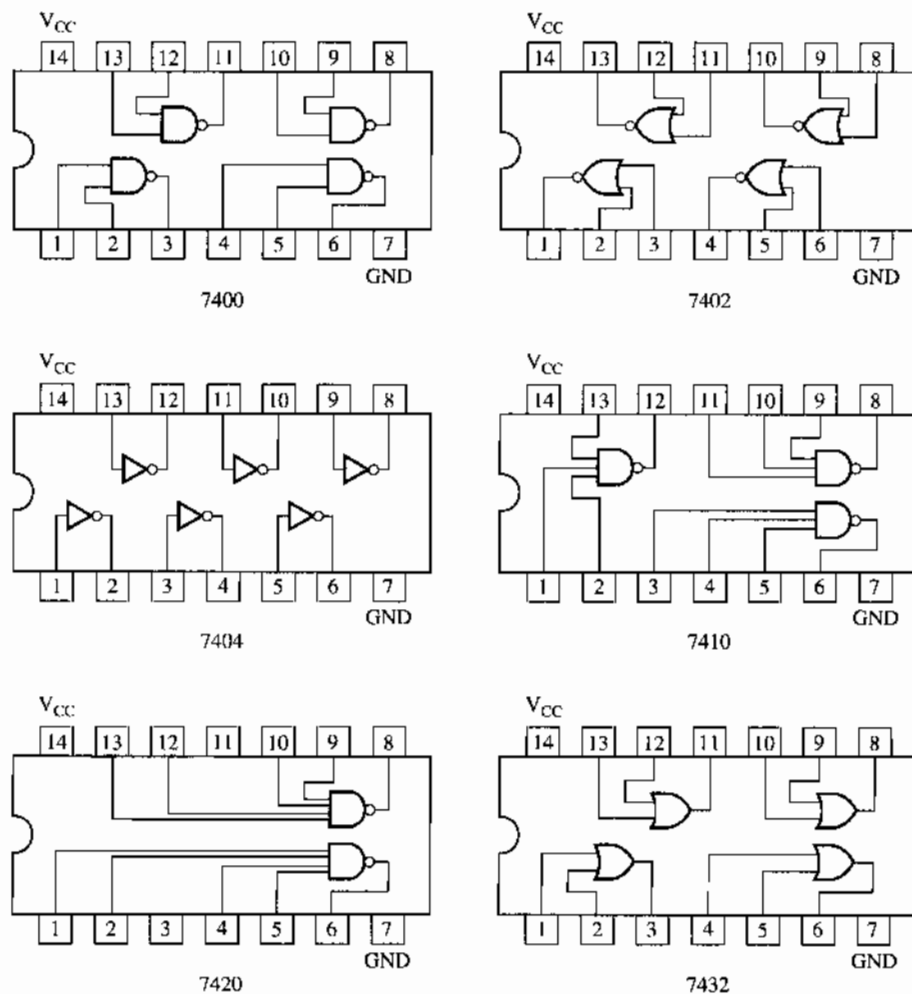


Figura 23. Circuitos integrados SSI característicos.

Confiabilidad. Se emplean técnicas de manufactura y prueba muy refinadas en la producción de los CI. Esto, junto con el hecho de que todos los componentes y sus interconexiones se fabrican al mismo tiempo en el proceso inicial, ofrece a los circuitos integrados más confiabilidad. Por la misma razón, puesto que se fabrican al mismo tiempo copias múltiples del mismo tipo de componente (digamos transistores), es mucho más fácil obtener componentes "acoplados", tanto en términos de valores de parámetros y sensibilidad a la temperatura. Estos componentes acoplados se requieren a menudo en un diseño de circuito.

Requerimientos de potencia. El calor que se genera en circuitos eléctricos debe eliminarse para evitar que los circuitos se sobrecalienten. (En las primeras computadoras se generaba tanto calor que se necesitaba un cuarto especial para enfriarlas.) El problema se reduce en gran medida si el consumo de potencia del circuito es bajo. El requerimiento de gran reducción de potencia de los CI y su pequeño tamaño tienen por ello una relación simbiótica. Con una demanda de potencia inferior, los circuitos se pueden empaquetar más densamente sin temor al sobrecalentamiento.

Costo. El proceso de fabricación de un solo transistor no es tan diferente del correspondiente a fabricar una compuerta o un circuito integrado SSI; es posible utilizar la misma pa-

tilla como punto de partida. El número de detalles de las operaciones de manufactura que se van a ejecutar es similar.

El costo de empaquetamiento de un solo transistor es apenas marginalmente menor que el costo de empaquetamiento de un CI más complicado. Las mismas consideraciones se aplican a dos CI diferentes con diferentes números de compuertas: el costo de CI con 30-40 compuertas es apenas ligeramente inferior que el de uno con 80-90 compuertas. En consecuencia, el costo de un CI no es directamente proporcional al número de componentes individuales incorporados en un paquete o cápsula; las etapas en el proceso de manufactura cuestan apenas marginalmente más si el número de compuertas en el paquete se duplica, por ejemplo.

Economía de diseño

Cuando los circuitos lógicos se constrúan con compuertas discretas ("primitivas"), los costos podrían reducirse al disminuir el número de compuertas requeridas para efectuar una función particular. Los procedimientos para determinar estos circuitos mínimos se volvieron altamente desarrollados. Estos procedimientos de minimización siguen desempeñando un papel, especialmente en conexión con el diseño lógico utilizando lo que se denomina *dispositivos lógicos programables* (PLD), como veremos en los capítulos 4 y 8. En consecuencia, les prestaremos cierta atención en el siguiente capítulo.

El costo de un circuito diseñado con CI SSI, sin embargo, no sólo depende del conteo de compuertas sino también del conteo de empaques de CI.²² Así, agregar una compuerta o dos a un diseño quizá no tenga ningún costo adicional si los paquetes SSI con compuertas sin usar están realmente presentes en el diseño. O supóngase que una tarea particular se alcanza con un circuito de 25 compuertas en dos paquetes SSI, y se dispone de un paquete de 40 compuertas (con compuertas que tienen el número apropiado de entradas); podría ser menos costoso utilizar un solo paquete en el diseño y "desperdiciar" las compuertas sin uso en lugar de recurrir a los dos paquetes con menor número de compuertas.

En algunos casos otra parte del circuito que se va a diseñar quizá requiera de unas cuantas compuertas del tipo que se dejó sin uso en el diseño que acaba de mencionarse. Estas compuertas previamente "desperdiciadas" se podrían usar ahora para este propósito, sin ningún costo adicional de hardware excepto por el correspondiente a la realización de las conexiones de las terminales, algo que se requeriría de cualquier manera. En todo caso, la economía del diseño ya no depende del conteo de compuertas, sino del conteo de paquetes de circuito integrado y del número de conexiones que se tiene que efectuar en la construcción del circuito.

Una forma en la que se economiza en el conteo de paquetes, por ejemplo, consiste en evitar utilizar inversores (el 74LS04s, por mencionar un caso). En vez de eso, se usa una fracción disponible de un paquete NAND, NOR o XOR. Considere las siguientes expresiones:

$$A' = (A + A + \dots + A)' = (A + 0 + \dots + 0)' \quad (25)$$

$$A' = (A \cdot A \cdot \dots \cdot A)' = (A \cdot 1 \cdot \dots \cdot 1)' \quad (26)$$

$$A' = A \oplus 1 \quad (27)$$

De (26) advertimos que el complemento de una variable es la salida de una compuerta NAND con cualquier número de entradas conectadas a esa variable en su totalidad, o una entrada conectada a esa variable y el resto conectada al voltaje alto (lógica 1, suponiendo lógica positiva). Así, si un cuarto de un 74LS00 (o un tercio de un 74LS10) está sin usar en un paquete cuyo resto ya

²² En la práctica, múltiples CI en un diseño se montan sobre una *tarjeta de circuito impreso* (TCI). De modo que el diseño económico no sólo implica el conteo de CI sino el de TCI también. Tales asuntos prácticos se aprenden con facilidad cuando nos adentramos en la práctica, lo cual no es nuestro propósito aquí.

forma parte de un circuito, es posible utilizar esta fracción del paquete para implementar un inversor sin un costo adicional de hardware.

Ejercicio 17. De manera similar, interprete las expresiones en (25) y (27) para implementar un inversor.

Como veremos en el capítulo 4, muchas unidades de circuito combinatorios que efectúan algunas funciones bastante complejas se han podido conseguir desde hace tiempo en los circuitos integrados MSI. Ya no existe ninguna necesidad de diseñar tales funciones mediante la combinación de compuertas individuales en paquetes SSI.

CI de aplicación específica

Los circuitos integrados hasta ahora han sido de propósito general. Los CI SSI ilustrados en la figura 23, por ejemplo, los CI MSI en los cuales se ponen en práctica las aplicaciones que se describen en el capítulo 4, o incluso los CI LSI que albergan unidades mayores pueden utilizarse en el diseño de una amplia variedad de aplicaciones. Un desarrollo más reciente es la posibilidad de diseñar un CI para una aplicación particular.

Si únicamente unos cuantos de los CI resultantes se utilizaran alguna vez, entonces no tendría sentido económico dedicar los recursos de diseño de ingeniería requeridos. Por otro lado, si es grande el número de copias necesarias, quizá resulte económicamente factible diseñar un circuito integrado completo justo para esta aplicación particular. No debe sorprender que este tipo de CI reciba el nombre de *circuito integrado de aplicación específica (ASIC)*.

Dos tipos de costos se asocian con un ASIC. Una vez que la unidad se diseña y depura, habrá costos de producción. Desde luego, el diseño real y la propia depuración acarrearán un costo sustancial, aunque esto no dependerá de los costos de producción posteriores; éste es un *costo de ingeniería no recurrente (NRE)*. ¿Por qué emprender una empresa que requiere costos NRE elevados a no ser que exista alguna ventaja de costo o de desempeño? En realidad, los ASIC ofrecen ventajas: por lo general, el sistema utiliza menor número de CI, tiene menor tamaño físico y, en consecuencia, inferior consumo de potencia y retardos de tiempo más pequeños, y por ello productos de retardo-potencia de menor tamaño. Con lo ya expuesto, concluiremos con este tema.

11 LÓGICA ALAMBRADA

Algunas veces resulta conveniente efectuar funciones lógicas utilizando configuraciones y conexiones de circuito especiales en vez de compuertas lógicas (por ejemplo, conectando simplemente dos o más alambres juntos). En esta sección se describen dos tipos de compuerta que permiten el uso de estas configuraciones y conexiones especiales. Para el lector probablemente no es obvia en este punto la utilidad de estas compuertas, pero lo será cuando lleguemos al capítulo 4. Las consideraremos aquí debido a que su implementación depende de los circuitos de transistores que se describieron en la sección anterior.

Compuertas lógicas de tres estados (alta impedancia)

Considere el circuito equivalente para la etapa de salida de una compuerta CMOS como se muestra en la figura 16. En la compuerta CMOS convencional las entradas lógicas a la compuerta controlan el estado de los interruptores (los transistores de salida). Por ejemplo, en una compuerta NAND de dos entradas el interruptor entre la salida y tierra se cierra si ambas entradas son 1 lógico; en otro caso, se cierra el interruptor entre la salida y el suministro eléctrico. Los dos in-

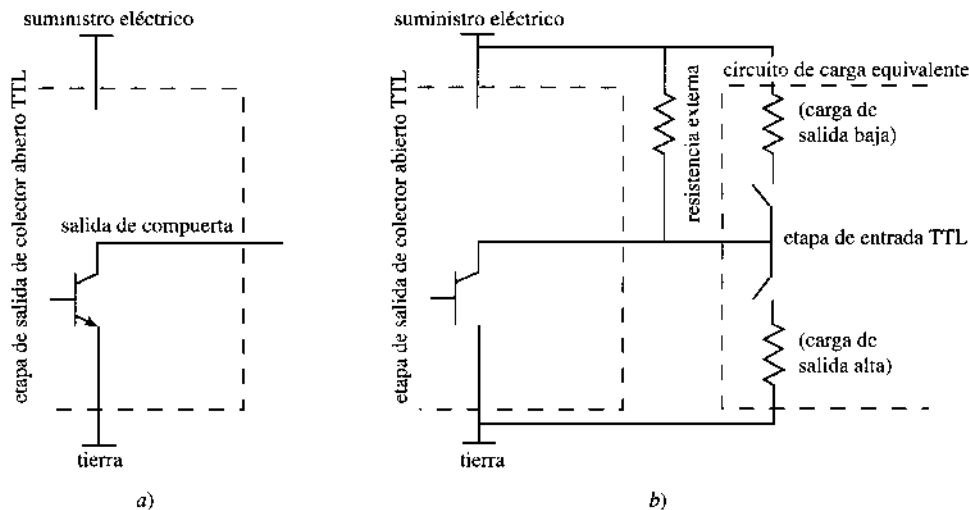


Figura 24. a) La etapa de salida de una compuerta TTL de colector abierto.
b) La configuración de circuito para la compuerta de colector abierto.

terruptores nunca se cierran al mismo tiempo, ya que esto originaría un nivel de voltaje de salida que no corresponde con un 1 lógico o con un 0 lógico. En una compuerta convencional la salida siempre presenta uno de los niveles lógicos válidos.

¿Existe alguna ventaja al abrir los dos interruptores al mismo tiempo? Si ambos interruptores se abren, entonces la salida ya no se acciona en lo absoluto; esto es, se queda flotando. En este estado la salida tiene una impedancia muy alta, por lo que se le denomina el estado de *alta impedancia* (o Z alta). Se le puede dar a una compuerta la capacidad de entrar a este tercer estado añadiendo una entrada adicional que, independientemente de las entradas lógicas, pueda abrir ambos interruptores de salida. Esta entrada adicional a menudo recibe el nombre de entrada *habilitadora*. Cuando está activa, la compuerta se comporta como una convencional, pero cuando está inactiva se encuentra en el estado de alta impedancia. Una compuerta con estas características se denomina de alta impedancia o de tres estados.

Es posible conectar las salidas de dos compuertas de tres estados en forma directa entre sí en el caso de que éstas nunca se hayan habilitado al mismo tiempo. La operación de una compuerta lógica TTL de tres estados es similar.

Ejercicio 18. Explique el comportamiento de dos compuertas lógicas convencionales si sus salidas se conectan directamente entre sí. (¿Hay diferencia si las compuertas son TTL o CMOS?) •

Compuertas lógicas de colector abierto y de drenaje abierto

Suponga que se construye una nueva compuerta a partir de una TTL convencional (figura 18) en la cual se elimina uno de los transistores en la salida. La nueva compuerta tendría la capacidad de llevar su salida hacia un estado lógico pero no hacia el otro. Vamos a eliminar el transistor que activa la salida en estado alto. El nuevo circuito se muestra en la figura 24a. ¿Es útil dicho circuito? Tal vez, pero sólo si hay alguna manera de manejar la salida hacia el estado alto cuando sea necesario. Esto puede conseguirse utilizando una resistencia externa como se muestra en la figura 24b.

Cuando las entradas de la compuerta son tales que la salida se pone en el estado bajo, ésta se activa en el estado bajo a través de la salida de la compuerta TTL; la resistencia actúa como una carga adicional en paralelo con las compuertas de carga. El valor de resistencia no debe ser tan pequeño como para sobrecargar la compuerta. Cuando las entradas son tales que la salida debe ser llevada al estado alto, la salida ya no la activa la compuerta TTL; la resistencia externa proporciona una trayectoria para que en vez de eso la corriente accione las compuertas de carga.

Advierta que cuando la salida alcanza el estado alto, la resistencia externa se encuentra en serie con las etapas de entrada equivalentes de las compuertas de carga; de esta forma, su valor debe elegirse de manera tal que se alcance un voltaje de salida correspondiente a un nivel lógico válido. Este tipo se conoce como compuerta de *colector abierto* debido a que el colector del transistor de salida es un circuito abierto sin ninguna conexión interna en el CI. Es posible construir una compuerta similar en CMOS a la cual se le denomina compuerta de *drenaje abierto* en virtud de que, dentro del CI, no se conecta la terminal de drenaje del transistor de salida.

La utilidad de este circuito no es obvia. Además, necesitamos más componentes que con los que empezamos (una resistencia externa). Sin embargo, si conectamos las salidas a dos o más de tales compuertas, contamos con un circuito que proporciona siempre un nivel lógico válido siempre y cuando la resistencia externa tenga un valor apropiado.

Si una o más de las compuertas provocan un nivel de salida baja, entonces la resistencia externa actúa como una carga adicional. Si ninguna de las compuertas da lugar a la salida baja, entonces la resistencia externa proporciona una trayectoria conductiva para la corriente y pone la salida en el estado alto. Por esta razón, la conexión se denomina *AND alambrada*. Si una o más de las salidas alambradas en conjunto corresponden a un 0 lógico, entonces, considerando la lógica positiva, lo mismo ocurre con la señal; en otro caso, ésta corresponde a un 1 lógico.

RESUMEN Y REPASO DEL CAPÍTULO

En este capítulo se presentó el álgebra booleana y los resultados que siguen de su uso. Se presentaron también los dispositivos fundamentales, denominados compuertas, sobre los cuales se basa el diseño lógico. Se incluyeron los siguientes temas:

- Postulados de Huntington
- Álgebra booleana
- Involución
 - Idempotencia
 - Absorción
 - Ley asociativa
 - Consenso
- Ley de De Morgan
- Operaciones de conmutación: AND, OR, NOT, NAND, NOR, XOR, XNOR
- Expresiones de conmutación
 - Forma de suma de productos
 - Forma de producto de sumas
 - Forma canónica
- Minitérminos y maxitérminos
- Funciones de conmutación
- Teorema de expansión de Shannon: s de p y p de s
- Operaciones universales
- Compuertas lógicas
- Formas equivalentes de compuertas NAND y compuertas NOR

- Lógica positiva, negativa y combinada
- Familias de compuertas lógicas: TTL, CMOS
- Propiedades de compuerta
 - Factor de carga de salida
 - Factor de carga de entrada
 - Consumo de potencia
 - Retardo de la propagación
 - Velocidad
 - Margen de ruido
- Circuitos integrados: SSI, MSI, LSI, VLSI, ASIC
- Lógica alambrada
 - Compuertas de tercer estado
 - Compuertas de colector abierto
 - Compuertas de drenaje abierto

PROBLEMAS

1. Demuestre los siguientes teoremas del álgebra booleana.
 - a. Los elementos identidad son distintos, esto es, no son los mismos.
 - b. Los elementos identidad son únicos, no hay otros.
 - c. El inverso de un elemento es único, esto es, no hay dos inversos diferentes.
 - d. El complemento de un elemento es único. A partir de esto, demuestre que $(x')' = x$.
2. Emplee la ley distributiva y otras leyes del álgebra de conmutación para poner cada una de las expresiones siguientes en la forma más simple de suma de productos. (Es de utilidad recordar estos resultados.)
 - a. $f_1 = (A + B)(A + C)$
 - b. $f_2 = (A + B)(A' + C)$
3. Complete los detalles de la demostración de la ley asociativa (teorema 6) descrita en el texto.
4. Verifique el teorema de consenso mediante *inducción perfecta*, esto es, demostrando que es cierta para todos los valores posibles de las variables.
5. Demuestre que no existe un álgebra booleana de tres elementos distintos, digamos $\{0, 1, 2\}$. (Si todos los postulados de Huntington se satisficieran, entonces, *existiría*.)
6. El siguiente conjunto de cuatro elementos se está considerando como los elementos de una álgebra booleana: $\{0, 1, a, b\}$. De los elementos identidad que cumplen el postulado, dos van a ser 0 y 1. Complete las tablas para (+) y (·) en la figura P6 de manera que el sistema algebraico definido por estas tablas será un álgebra booleana. Encuentre el complemento de cada elemento, y confirme que se satisfacen todos los postulados de Huntington.
7. Las operaciones de un sistema algebraico de cuatro elementos se dan en la figura P7. Determine si este sistema es un álgebra booleana. Si es así, ¿cuáles son los elementos identidad?
8. En un restaurante de comida natural, se ofrece fruta como postre pero sólo en ciertas combinaciones. Una elección es durazno o manzanas o ambos. Otra corresponde a cerezas o manzanas o ninguna de ellas. Una tercera elección son duraznos, aunque si éstos se eligen, también deben aceptarse plátanos. Defina las variables booleanas para la totalidad de estas frutas y escriba una expresión lógica que especifique la fruta disponible para el postre. Después simplifique la expresión.
9. Escriba una expresión lógica que represente la siguiente proposición: la corriente de colector en un transistor bipolar es proporcional al voltaje base-emisor v_{BE} . Siempre que el transistor no esté saturado ni en corte.

(+)	0	a	b	1
0				
a				
b				
1				

a)

(•)	0	a	b	1
0				
a				
b				
1				

b)

Figura P6

x	x'
a	c
b	d
c	a
d	b

a)

(+)	a	b	c	d
a	a	a	a	a
b	a	b	b	a
c	a	b	c	d
d	a	a	d	d

b)

(•)	a	b	c	d
a	a	b	c	d
b	b	b	c	c
c	c	c	c	c
d	d	c	c	d

c)

Figura P7

10. Construya una tabla de verdad para cada función representada por las siguientes expresiones.

a. $E = (A' + B)(A' + B' + C)$

b. $E = (((A' + B) + C)' + A)'$

c. $E = xz' + yz + xy'$

11. Realice la demostración del teorema de expansión de Shannon dado como (6) en el texto.

12. Utilice una o ambas de las leyes distributivas (en forma repetida si es necesario) para poner cada una de las siguientes expresiones en la forma de producto de sumas.

a. $f_1 = x + wyz'$

b. $f_2 = AC + B'D'$

c. $f_3 = xy' + x'y$

d. $f_4 = xy' + wuv + xz$

e. $f_5 = BC' + AD'E$

f. $f_6 = ABC' + ACD' + AB'D + BC'D$

13. Construya una tabla de verdad para cada función representada por las siguientes expresiones.

a. $E = xy + (x + z')(x + y' + z') + xy'z$

b. $E = (w + x'z' + y)(y' + z') + wxy'z$

c. $E = (AB'C + BC'D) + AB'D + BCD'$

14. Aplique la ley de De Morgan (en forma repetida si es necesario, pero sin ninguna manipulación simplificatoria) para determinar el complemento de cada una de las siguientes expresiones.

a. $f = AB(C + D') + A'C'(BD' + B'D)$

b. $f = [AC' + (B' + D)(A + C')][BC + A'D(CE' + A)]$

15. Verifique la siguiente expresión utilizando las reglas del álgebra booleana.

$$x'y + y'z + yz' = x'z + y'z + yz'$$

16. Utilizando las reglas del álgebra booleana, simplifique las expresiones que siguen hasta el menor número total de literales.

a. $f = AB' + ABC + AC'D$

b. $f = wyz + xy + xz' + yz$

c. $f = B + AD + BC + [B + A(C + D)]'$

17. Recurra al álgebra de conmutación para simplificar lo más posible las siguientes expresiones.

a. $xyz' + xy'z' + x'y$

b. $(wx')'(w + y)(x'y'z')'$

- c. $x'(y + wy'z') + x'y'(w'z' + z)$
 c. $(w + x)(w' + x + yz')(w + y')$
18. a. Realice las operaciones de conmutación adecuadas en las siguientes expresiones para llegar a formas no canónicas de suma de productos. Después reponga las variables faltantes para convertirlas a la forma canónica.
 b. En cada caso, utilizando la ley distributiva y otras leyes booleanas, convierta la forma no canónica de suma de productos a la forma de producto de sumas.
 c. En cada caso, reponga las variables faltantes a las formas determinadas en la parte a para convertirlas a la forma canónica.
 d. En cada caso, construya la tabla de verdad y confirme la forma canónica de suma de productos.
 e. En cada caso, aplicando la ley de De Morgan a los complementos de los minitérminos ausentes de la forma canónica de suma de productos, encuentre la forma canónica de producto de sumas.

$$E_1 = (x + y')(y + z')$$

$$E_2 = (x'y + x'z)'$$

$$E_3 = (x + yz')(y + xz')$$

$$E_4 = (B + D')(A + D')(B' + D')$$

$$E_5 = (AB)'(B + C'D)(A + D')$$

19. Utilizando álgebra de conmutación demuestre las siguientes relaciones.

a. $x' \oplus y = x \oplus y'$

b. $x' \oplus y' = x \oplus y$

c. $x'y \oplus xy' = x \oplus y$

20. Utilice las leyes apropiadas del álgebra de conmutación para convertir la expresión $(x \oplus y)(x \oplus z)$ a la forma $A \oplus B$; especifique A y B en términos de x , y , z .
21. a. Expresé la función OR exclusiva, $x \oplus y$ en términos únicamente de funciones NAND.
 b. Expresé $x \oplus y$ en términos únicamente de funciones NOR.
22. Compruebe que la operación OR exclusiva es asociativa. Esto es, $(x \oplus y) \oplus z = x \oplus (y \oplus z)$.
23. Demuestre que $\{NOR\}$ es un conjunto de operaciones universal.
24. La función *implicación* $f = (x \Rightarrow y)$ es el enunciado "Si x es verdadera, entonces y es verdadera". Una expresión de conmutación para la función de implicación corresponde a $f = x' + y$. Esta función se puede implementar por medio de una compuerta OR de dos entradas en la cual una de las líneas de entrada tiene una burbuja. Ésta podría denominarse una compuerta *de implicación*.
- a. Construya una tabla de verdad para la función implicación. ¿Es incomprendible alguno de los renglones?
 b. Demuestre que la función implicación constituye un conjunto universal; esto es, cualquier expresión de conmutación puede representarse en términos únicamente de funciones implicación.
 c. Escriba la siguiente expresión únicamente en términos de funciones implicación: $f = AC + BC'$.
25. Cada una de las siguientes expresiones incluye las tres operaciones booleanas. Utilizando los teoremas booleanos apropiados, convierta cada expresión a una que incluya:

a. Sólo operaciones AND y NOT

b. Sólo operaciones OR y NOT

c. Sólo operaciones NAND

d. Sólo operaciones NOR

$$E_1 = AB' + AC + B'C$$

$$E_2 = (x' + y')(y' + z)(y + z')$$

$$E_3 = xy' + (y + z')(x + z)$$

26. a. Utilice el resultado de la figura 2 para convertir un circuito AND-OR de dos niveles que consista en dos compuertas AND de entrada doble seguidas por una compuerta OR dentro de un circuito todo NAND.

- b. Utilice el resultado del ejercicio 15 del texto para convertir un circuito OR-AND de dos niveles consistente en dos compuertas OR de entrada doble seguidas por una compuerta AND y un circuito todo NOR.

27. Muestre que la operación OR en la primera forma del teorema de Shannon puede sustituirse por una OR exclusiva.
 28. Determine la relación entre la función OR exclusiva y su dual.
 29. El objetivo de este problema es convertir un tipo de compuerta de dos entradas en un tipo diferente de compuerta, pero con entradas invertidas. Considere que las entradas son A y B . Utilice el teorema booleano apropiado a fin de satisfacer este objetivo para los siguientes tipos de compuerta:

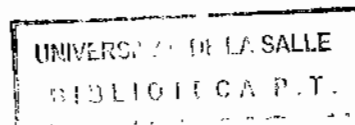
AND: AB

OR: $A + B$

NAND: $(AB)'$

NOR: $(A + B)'$

30. Una compuerta OR tiene dos entradas A y B . B se obtiene de A después de que A recorre tres inversores consecutivos. Suponga que cada inversor tiene un retardo de $1 \mu s$. Suponga que A ha sido 1 por algún tiempo y que después se reduce a 0. Dibuje la forma de onda de la salida resultante.
 31. Una carga estándar para una compuerta LS-TTL corresponde a 3mA de corriente; tal compuerta tiene la posibilidad de activar tres cargas estándar del mismo tipo de compuerta. Una carga estándar para una compuerta S-TTL es igual a 7 mA. ¿Cuántas compuertas S-TTL puede accionar una compuerta LS-TTL?
 32. Suponiendo que son compatibles los niveles de voltaje lógico-alto y lógico-bajo de dos subfamilias CMOS y TTL, ¿cuáles son las consecuencias de accionar una compuerta TTL con una compuerta CMOS y viceversa?
 33. El retardo intrínseco de una compuerta es de 0.5 ns y el retardo extrínseco corresponde a 0.2 ns por carga. ¿Cuál es la limitación del factor de carga de salida de esta compuerta, dado que su retardo no puede exceder 1.2 ns?
 34. ¿Es posible conectar las salidas de compuertas convencionales a las salidas de compuertas de colector abierto? En caso afirmativo, muestre un circuito de este tipo y describa su operación. Si no, explique por qué no pueden conectarse entre sí.
 35. ¿Cuál es la consecuencia de conectar juntas muchas salidas de tres estados? ¿Cuántas salidas de tres estados pueden conectarse sin afectar la funcionalidad propia de un circuito?
 36. ¿Cuál es la consecuencia de conectar entre sí muchas salidas de colector abierto? ¿Cuántas salidas de colector abierto es posible conectar sin afectar la funcionalidad propia de un circuito?
 37. ¿Las salidas de colector abierto y las compuertas de tres estados pueden conectarse entre sí? Explique por qué sí o por qué no.



Representación e implementación de funciones lógicas

En el capítulo anterior se abordó el álgebra de conmutación. Ahí encontramos que las entidades denominadas variables de conmutación se relacionan por medio de varias operaciones de conmutación con diferentes combinaciones de estas operaciones denominadas *expresiones de conmutación* o *lógicas*. Observamos que es posible representar una función de conmutación mediante una o más expresiones.

Supóngase que interpretamos las variables como señales. Entonces es posible interpretar una expresión de conmutación $E(x_1, x_2, \dots, x_n)$ como operaciones efectuadas sobre las señales de entrada x_i , originando una señal de salida acorde con la expresión. Como se describió brevemente en el capítulo anterior, la meta final es implementar las expresiones de conmutación mediante circuitos físicos y hardware. Puesto que una función de conmutación puede representarse mediante muchas expresiones diferentes, puede implementarse de muchas maneras diferentes en hardware. Algunos arreglos de hardware quizá sean en cierto modo más deseables que otros; es posible que resulten más simples, más convenientes, económicos, rápidos o menos demandantes de potencia. Examinaremos en detalle estos aspectos a partir del siguiente capítulo.

Este capítulo explorará métodos alternativos de expresión de funciones lógicas: geométricos, algebraicos y tabulares.

Investigaremos procedimientos para convertir expresiones en formas simples después de decidir primero el significado de "simple".

Analizaremos también la implementación de funciones lógicas por medio de compuertas primitivas en circuitos integrados a pequeña escala.

En los últimos capítulos se explicarán implementaciones más extensivas en circuitos MSI y LSI.

1 LISTAS DE MINITÉRMINOS Y MAXITÉRMINOS

Como se definió en el capítulo anterior, una función de conmutación de n variables es una asignación específica de valores binarios a cada una de las 2^n combinaciones de valores de las n variables.

De ese modo, una función de conmutación está completamente identificada por su tabla de verdad. Sin embargo, una tabla de verdad puede tener un gran número de renglones y columnas, —16 renglones y 5 columnas para 4 variables, por ejemplo. Lo que necesitamos es un procedimiento más simple para especificar una función de conmutación.

Código decimal	$x \ y \ z$	f	Minterm	Maxterm
0	0 0 0	0	$x'y'z'$	$x + y + z$
1	0 0 1	1	$x'y'z$	$x + y + z'$
2	0 1 0	0	$x'yz'$	$x + y' + z$
3	0 1 1	1	$x'yz$	$x + y' + z'$
4	1 0 0	1	$xy'z'$	$x' + y + z$
5	1 0 1	0	$xy'z$	$x' + y + z'$
6	1 1 0	0	xyz'	$x' + y' + z$
7	1 1 1	0	xyz	$x' + y' + z'$

Figura 1. Tabla de verdad para una función dada f .

La explicación se iniciará con el siguiente ejemplo de una función de conmutación representada por medio de una expresión de forma mezclada:

$$f(x, y, z) = (x' + y' z')(z + xy')$$

Ésta puede convertirse en una forma no canónica de suma de productos aplicando la ley distributiva y otras reglas booleanas; también es posible transformarla en la forma canónica. Lleve a cabo esta sugerencia y luego de eso verifique su resultado con el siguiente.

$$f = x'z + xy' z' = x'z(y + y') + xy' z' = x'y' z + x'yz + xy' z'$$

Cada término en la forma final es un minitérmino. Cada minitérmino es 1 para una combinación específica de valores de las variables. En consecuencia, la función tiene el valor 1 para tres combinaciones diferentes de valores de las variables: 001, 011 y 100, como se indica en la tabla de verdad de la figura 1. (Temporalmente ignore las últimas dos columnas, pues no son parte de la tabla de verdad; se agregaron porque así conviene más adelante.)

Listas de minitérminos y forma de suma de productos

Es claro que la tabla de verdad contiene gran cantidad de información redundante. Para identificar una función, no es necesario especificar tanto aquellos lugares donde la función es 1 como aquellos donde es 0; basta listar únicamente las combinaciones específicas de valores para los cuales la función corresponde a 1, o sólo los valores para los que la función es 0. Más simple aún es listar sus equivalentes decimales. Así, la función correspondiente a la tabla de verdad anterior puede identificarse simplemente como:

$$f(x, y, z) = \Sigma(1, 3, 4)$$

Dicha representación de una función se conoce como *lista de minitérminos*. El símbolo sumatoria significa que los minitérminos identificados por sus valores decimales se van a sumar. Cuando una expresión de conmutación se da en cualquier forma, una manera de determinar la lista de minitérminos consiste en construir primero la tabla de verdad; la lista de minitérminos puede escribirse entonces por inspección. Alternativamente, la expresión dada se convierte en la forma canónica de suma de productos; el número de minitérminos correspondiente a cada minitérmino se obtiene en ese caso convirtiendo a la forma decimal el número binario que representa a ese minitérmino. Por consiguiente, para cuatro variables, un minitérmino $ABC'D'$ toma el valor 1 para la combinación 1100 o decimal 12.

Desde luego, lo inverso también es posible. Dada una lista de minitérminos, es posible generar el minitérmino correspondiente a cada número de minitérmino. La suma de todos los minitérminos representa la forma canónica de suma de productos. Considere, por ejemplo, la siguiente lista de minitérminos:

$$f(A, B, C) = \Sigma(2, 3, 5, 7)$$

El minitérmino número 5 tiene el valor 1 para la combinación 101, que corresponde al minitérmino $AB'C$, y similarmente para los otros minitérminos.

Para referencia fácil, un minitérmino puede designarse m_i , donde i es el número de minitérmino (empleamos una m para el minitérmino). En el caso de una función de tres variables, por ejemplo, será $A'BC' = m_2$. En esta notación, la forma canónica de suma de productos de la función cuya lista de minitérminos se da en la expresión anterior, se escribe como:

$$f(A, B, C) = m_2 + m_3 + m_5 + m_7$$

Ejercicio 1. Se dio una tabla de verdad para una función f_1 en la figura 5 del capítulo 2. Indique tres representaciones de esta función: mediante la lista de minitérminos, a partir de la sumatoria de m_i apropiadas, y mediante la suma de minitérminos reales. (Las respuestas se proporcionan en los pies de página; no las vea hasta después de que realice el ejercicio.)

Respuesta¹

Listas de maxitérminos y forma de producto de sumas

En virtud del principio de dualidad, lo que se hizo en la subsección anterior en términos de sumas de productos, minitérminos y listas de minitérminos, también puede llevarse a cabo en términos de productos de sumas, maxitérminos y lo que recibe el nombre de *listas de maxitérminos*. Una función de conmutación puede identificarse verdaderamente determinando sus 1s. Igualmente lo anterior es posible especificando sus 0s. Esto es, las combinaciones de valores de las variables para las cuales el valor de la función es 0. En la tabla de verdad de la figura 1, por ejemplo, la función es 0 para cinco combinaciones de valores de variables, correspondiendo a los equivalentes decimales: 0, 2, 5, 6, 7. (No lea únicamente lo anterior; ¡compruébelo!) Los maxitérminos correspondientes a estos renglones de la tabla serán cero para los valores de las variables especificados. Por consiguiente, el *producto* de estos cinco maxitérminos, y no de otros, será 0. Simbólicamente, por tanto, es posible identificar la función como:

$$f(x, y, z) = \Pi(0, 2, 5, 6, 7)$$

Esta representación de una función es una *lista de maxitérminos*. El símbolo producto Π (pi) significa que los maxitérminos correspondientes se van a multiplicar. Con las expresiones para los maxitérminos dadas en la última columna en la tabla de verdad, la forma canónica de productos de sumas de la función se escribe de la manera siguiente:

$$f = (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z)(x' + y' + z')$$

Del mismo modo que un minitérmino se representa de manera simbólica mediante m_i , es posible representar simbólicamente un maxitérmino como M_i . (La mayúscula M se usará para el maxitérmino.) En esta notación, el producto de sumas canónico precedente puede escribirse como:

$$f = M_0 M_2 M_5 M_6 M_7$$

Ejercicio 2. Para la misma tabla de verdad que se usó en el ejercicio 1, proporcione tres representaciones de la función: una en términos de listas de maxitérminos, una simbólicamente utilizando un producto de M_i apropiadas, y la última multiplicando los maxitérminos reales.

Respuesta²

¹ $f_1 = \Sigma(1, 3, 6, 7) = m_1 + m_3 + m_6 + m_7 = x'y'z + x'yz' + xyz' + xyz$

² $f_1 = \Pi(0, 2, 4, 5) = M_0 M_2 M_4 M_5 = (x + y + z)(x + y' + z)(x' + y + z)(x' + y' + z')$

2 MAPAS LÓGICOS

La sección precedente demostró cómo pueden escribirse expresiones simbólicas simples —a saber, listas de minitérminos y maxitérminos que representan formas canónicas de suma de productos y de productos de sumas— para cualquier función de conmutación. Sin embargo, las formas canónicas desperdician muchas literales. La mayoría de las veces, los minitérminos se combinan mediante la eliminación de literales redundantes para producir expresiones con menor número de literales. Describiremos ahora una forma geométrica de representar una expresión de conmutación, como preámbulo de un método para resolver el problema de simplificación de expresiones de ese tipo.

Adyacencia lógica y adyacencia geométrica

Dada una expresión canónica de suma de productos, resultaría útil contar con un método infalible para obtener una expresión, equivalente a la canónica, con el número menor de términos. Si fuera posible que dos o más términos en una expresión se combinaran en un solo término, eso originaría ciertamente menor número de términos.

Considere dos minitérminos: $wxyz$ y $wxy'z$. Su suma es $wxyz + wxy'z = wxz(y + y') = wxz$, una expresión más simple que los dos minitérminos.

Necesitamos un nombre para describir cómo se relacionan dos de tales minitérminos. Con este fin, vamos a plantear la siguiente definición:

Una combinación de valores de variable de entrada se dice que es adyacente lógicamente a otra si las dos combinaciones difieren únicamente en una posición de bit.

En una expresión de tres variables, por ejemplo, las combinaciones de entrada (valores de verdad) 110 y 111 son lógicamente adyacentes puesto que difieren únicamente en una posición de bit (la última). El objetivo en este caso consiste en crear una representación geométrica (en cierta medida tabular) de una función. Descubrimos que tal representación es útil para llegar a una forma mínima de suma de productos o de producto de sumas.

Iniciaremos con una función de dos variables: x e y . El esqueleto de una representación geométrica se presenta en la figura 2a. Para completarlo, necesitamos especificar el orden de los valores x e y .

Hay cuatro posibilidades; vamos a asignar de manera arbitraria los valores x e y como se indica en la figura 2b. (Considere las otras posibilidades.) Para esta asignación de valores x e y , las combinaciones xy posibles se muestran dentro de las celdas en la figura 2c. Observe que las combinaciones de los valores x e y en cada renglón y en cada columna en esta figura son lógicamente adyacentes, como se definió antes. Sin embargo, advierta que las dos celdas en cada renglón y en cada columna son también físicamente, o mejor, *geoméricamente adyacentes* entre sí.

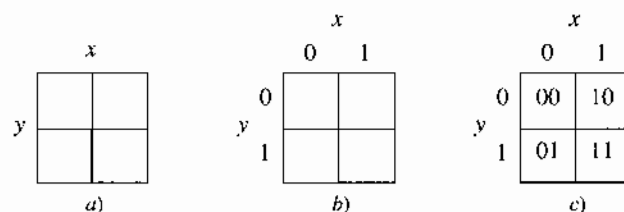


Figura 2. Representación geométrica de una función de dos variables
 a) estructura de mapa; b) valores x e y . c) Coordenadas de celda.

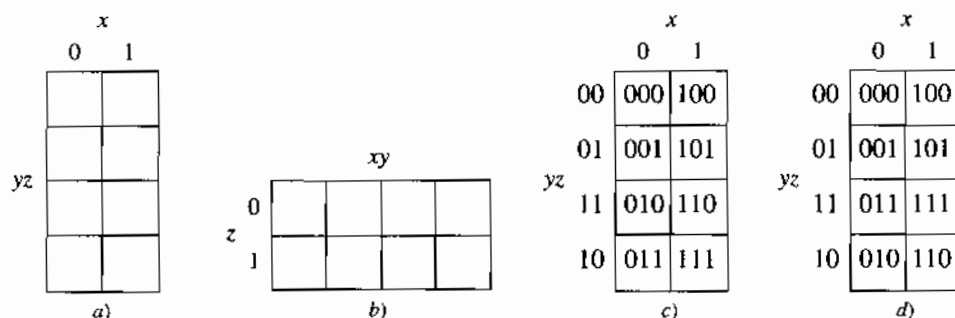


Figura 3. Estructura geométrica para el caso de tres variables.

Suponga ahora que el orden de valores de x o y , o de ambos, se invierte en este caso de dos variables (10 en lugar de 01 para x o y , o ambas). El mismo resultado es verdadero; esto es, que las celdas en cada renglón y en cada columna son adyacentes tanto geométrica como lógicamente. (Confirme esta aseveración.)

Puede resultar de este caso tan sencillo dos variables, no permitiendo una gran generalización, de modo que vamos a profundizar en el tema más a fondo. Las figuras 3a y 3b ilustran el arreglo posible de las variables cuando hay tres de ellas —una variable sobre la horizontal y dos variables sobre la vertical, o viceversa. ¿Cuál de estos arreglos que se usan es una cuestión de preferencia, y quién puede explicarla? En este libro usaremos la forma en la figura 3a, aunque algunos libros recurren a la indicada en la figura 3b. Si bien el orden de asignación de los valores de las variables para el eje de una sola variable (x) no es importante (como señalamos en el caso de dos variables), este orden marca una diferencia en las dos variables verticales en este caso, como advertiremos en lo que sigue.

Otra variación en el orden de la asignación de variables a los ejes horizontal y vertical es también posible. Hemos asignado la primera coordenada a las columnas y las últimas dos a los renglones. Lo opuesto también puede ocurrir y se usa en algunos libros: asignar las primeras dos coordenadas a los renglones y la última a la columna.

Suponga que los valores de dos variables (yz) en la figura 3a se asignan en código BCD, como se muestra en la figura 3c. Es claro que las dos celdas en cada renglón son geoméricamente adyacentes. Por inspección, vemos que también son *lógicamente adyacentes*. A continuación vamos a considerar las celdas adyacentes geoméricamente en cada columna; ¿éstas son también lógicamente adyacentes? (Estudie la figura 3c y llegue a una conclusión; sólo en ese caso vea la respuesta en el pie de página.)³

Esta desafortunada circunstancia surge de la elección del código BCD para ordenar los valores yz . Al pasar del segundo al tercer renglón, cambian los valores tanto de y como de z en cada columna. Al estudiar las entradas en las celdas, advertimos que si se intercambian los valores yz en el tercero y cuarto renglones (10 con 11), entonces las celdas geoméricamente adyacentes en cada columna ¡serán también lógicamente adyacentes! El resultado al efectuar lo anterior se muestra en la figura 3d. Indique el código en el cual los valores yz se expresan ahora.

Respuesta⁴

Es una práctica común identificar las celdas no por su representación binaria, como se hizo en esta figura, sino por su equivalente decimal. Los numerales se escriben en tipo pequeño en

³ En cada columna, dos celdas geoméricamente adyacentes son también lógicamente adyacentes, con excepción de aquellas en el segundo y el tercer renglón.

⁴ Código Gray.

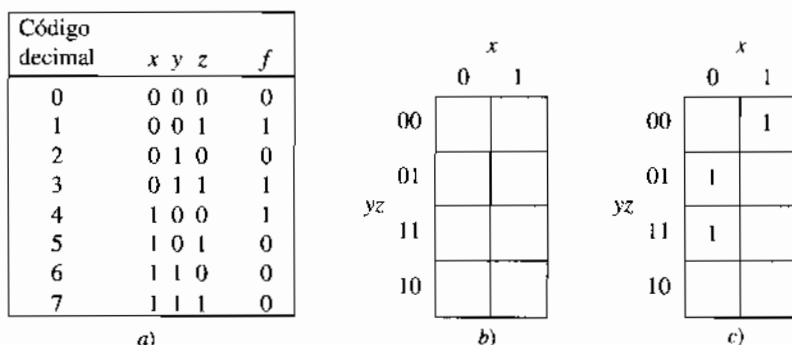


Figura 4. a) Tabla de verdad, b) estructura geométrica y c) representación de $f = \Sigma(1, 3, 4)$.

una esquina de la celda debido a que se efectuarán otras entradas en las celdas para describir funciones específicas. Estos numerales son exactamente los números de minitérminos. Quizá usted deba hacer lo anterior por cuenta propia hasta que se vuelva una cuestión secundaria trabajar con equivalentes binarios.

Es posible observar otro rasgo interesante del caso de tres variables en la figura 3d. Advierta en esta figura que en cada columna, la celda de hasta arriba es adyacente lógicamente a la que se encuentra hasta abajo, aunque las dos no parecen ser geoméricamente adyacentes.

Esta anomalía puede explicarse. Piense en reproducir este mapa sobre una hoja de papel cuadrada (utilizando la hoja completa) y enrollando ésta verticalmente en un cilindro, con el mapa en el exterior. (Utilice cinta adhesiva, si desea, para mantener unido el cilindro de manera que pueda efectuar con mayor facilidad sus observaciones.) Lo que *era* el renglón de abajo será ahora adyacente geoméricamente a lo que *era* el renglón de arriba. Ahora bien, para cada valor de x (esto es, en cada columna), las celdas geoméricamente adyacentes ¡también serán lógicamente adyacentes! El resultado de este argumento se resume de la manera siguiente:

El orden de las combinaciones de variables se elige de manera que dos celdas cualesquiera, geoméricamente adyacentes sean también lógicamente adyacentes.

Considere, por ejemplo, la celda 011 (decimal 3) en la figura 3d. Las celdas lógicamente adyacentes se obtienen sustituyendo, uno a la vez, cada valor de bit por su complemento; éstos son:

111 (decimal 7)
001 (decimal 1)
010 (decimal 2)

A partir de la figura, verifique que estas celdas son en realidad también geoméricamente adyacentes a 011.

La estructura geométrica en la cual dos celdas cualesquiera, geoméricamente adyacentes son también lógicamente adyacentes, se conoce como *mapa lógico*.⁵ Estamos listos ahora para lograr el objetivo de determinar un método para simplificar expresiones lógicas, llegando a formas mínimas de suma de productos o de producto de sumas.

La tabla de verdad de función de tres variables se presentó en la figura 1. Se muestra de nuevo en la figura 4a, y la estructura del mapa lógico para tres variables se ilustra en la figura 4b, utilizando el código Gray para las variables verticales. El mapa lógico de la función misma está

⁵ A menudo se denomina *mapa de Karnaugh* (abreviado mapa K), en honor a Maurice Karnaugh, quien fue el primero que lo propuso en 1953. E. W. Veitch había propuesto una forma un poco diferente un año antes. Hemos elegido darle un nombre de acuerdo con la función y no con la historia, aunque podríamos equivocarnos de vez en cuando y llamarlo mapa K.

		wx			
		00	01	11	10
yz	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

a)

		wx			
		00	01	11	10
yz	00				
	01	1			1
	11	1	1	1	1
	10			1	

a)

Figura 5. Mapa lógico de cuatro variables de $f = \Sigma(1, 3, 7, 9, 11, 14, 15)$.

en la figura 4c; hay 1s sólo en tres celdas que corresponden a los minitérminos. Esto es, cada minitérmino de una función corresponde a una celda del mapa para la cual la función es 1. Una sola celda es la más pequeña unidad que conforma al mapa.

Luego de esto introduciremos en cada celda el valor de la función dado en la tabla de verdad. ¿Realmente es necesario incorporar los valores tanto de 1 como de 0? Si sólo se introducen los 1s, por ejemplo, aquellas celdas en las cuales no hay entrada deben llevar un 0. En el mapa lógico de la figura 4c, sólo se introdujeron los 1s. Cada minitérmino de la función representada por este mapa corresponde a una celda del mapa que lleva un 1. Una sola celda es la unidad más pequeña que conforma al mapa; constituye la entidad mínima que puede mostrarse en él. ¡Ésta es la causa de la designación *minitérmino*! Hay una explicación similar para la designación *maxitérmino*.

Ejercicio 3. Utilizando un mapa lógico de tres variables, proporcione una explicación que haga que la designación “maxitérmino” sea plausible para una celda que contiene un 0 de una función. ♦

La herramienta del mapa lógico resulta muy útil. Nuestro siguiente paso debe ser extenderlo a funciones de más variables. Consideraremos primero cuatro variables; en este caso esperaríamos que un mapa tenga dos variables en cada eje. Con las variables w, x, y, z , supondríamos un mapa lógico de 2 por 2 (utilizando, digamos, wx para identificar las columnas e yz para identificar los renglones). Puesto que encontramos que el código Gray era antes el mejor, parece razonable asignar los valores tanto wx como yz de acuerdo con dicho código; si esto no funciona por alguna razón, podemos siempre probar con algún otro.

EJEMPLO 1

A continuación se presenta la lista de minitérminos de una función de cuatro variables. El objetivo es construir un mapa lógico para esta función.

$$f(w, x, y, z) = \Sigma(1, 3, 7, 9, 11, 14, 15)$$

La estructura de un mapa de cuatro variables utilizando el código Gray para asignar valores sobre ambos ejes se muestra en la figura 5a. El mapa para la función dada se construye insertando el valor 1 en cada celda correspondiendo al número de minitérmino de la función. Esto se muestra en la figura 5b. Confirme las entradas en este mapa. ♦

		$v = 0$				$v = 1$			
		wx				wx			
		00	01	11	10	10	11	01	00
yz	00	0	4	12	8	16	20	28	24
	01	1	5	13	9	17	21	29	25
	11	3	7	15	11	19	23	31	27
	10	2	6	14	10	18	22	30	26

Figura 6. Estructura del mapa de cinco variables.

Vamos a considerar el dibujo del mapa en la figura 5a sobre una hoja de papel y enrollar ésta desde la parte superior a la inferior en forma de cilindro, con el mapa en el exterior. (Será benéfico que usted efectúe esta tarea.) Las celdas en el renglón inferior son ahora adyacentes a las correspondientes en el renglón superior, tanto geométrica como lógicamente. Desenrolle ahora ese cilindro y cree otro, esta vez enrollándolo verticalmente, colocando la frontera derecha geoméricamente adyacente a la frontera izquierda. Confirme que las celdas en la columna izquierda son también lógicamente adyacentes a las celdas correspondientes en la columna derecha. Esto es, las dos celdas al final de cada renglón en el mapa plano son lógicamente adyacentes, al igual que aquellas en la parte superior e inferior de cada columna. Esto es lo que le da al mapa lógico su valor.

Después de que usted forma uno de los cilindros señalados en el párrafo anterior, otra torsión adicional proporcionará igualmente mayor conocimiento. Imagine que tuerce el cilindro para obtener un toroide, donde la sección transversal en un extremo resulta geoméricamente adyacente a la del otro. (Usted no puede hacer completamente esto con buenos resultados a menos que su hoja sea elástica.) Se observa que las cuatro celdas en las esquinas del mapa plano son geoméricamente adyacentes en pares, y que son también lógicamente adyacentes en pares. En la sección subsecuente se explicará más acerca de este aspecto.

Advierta que cualquier celda interna en un mapa de cuatro variables tiene un lado común con otras cuatro celdas, las únicas a las cuales es geoméricamente adyacente. Utilizando las construcciones del cilindro y el toroide descritas en el párrafo anterior, demuestre por cuenta propia que lo mismo es cierto para las celdas alrededor del perímetro del mapa. La generalización corresponde a que cualquier celda en cualquier parte en el mapa de tres variables es adyacente a las otras tres celdas.

La estructura del mapa de cinco variables se da en la figura 6. Consiste en dos mapas de cuatro variables uno al lado del otro, con la primera variable, v , adquiriendo el valor 0 en la mitad izquierda y el valor 1 en la mitad derecha del mapa. Note el orden de los valores wx para $v = 1$ comparado con aquéllos para $v = 0$. Como ejercicio, confirme (enrollando de nuevo el mapa en un cilindro) que las celdas en la columna más a la izquierda para $v = 0$ son lógicamente adyacentes a las celdas correspondientes en la columna más a la derecha para $v = 1$, como lo son aquéllas en el renglón superior y en el inferior. Advierta también en este caso que el código en el cual se ordenan las cinco variables es el Gray.⁶

Ejercicio 4. Considere la celda en un mapa lógico de cinco variables identificado por el minitérmino 01011. Localice esta celda sobre un mapa de cinco variables. Especifique los números de minitérmino de las cinco celdas a los cuales ésta es lógicamente adyacente y localícelos sobre el mapa.

Respuesta⁷

⁶ Maurice Karnaugh tuvo que *concebir* la idea de que si los valores de las variables se listaran en el orden del código Gray en un mapa de n variables, entonces cada celda sería lógicamente adyacente a otras n .

⁷ Obténgalas sustituyendo, uno a la vez, cada bit por su complemento.

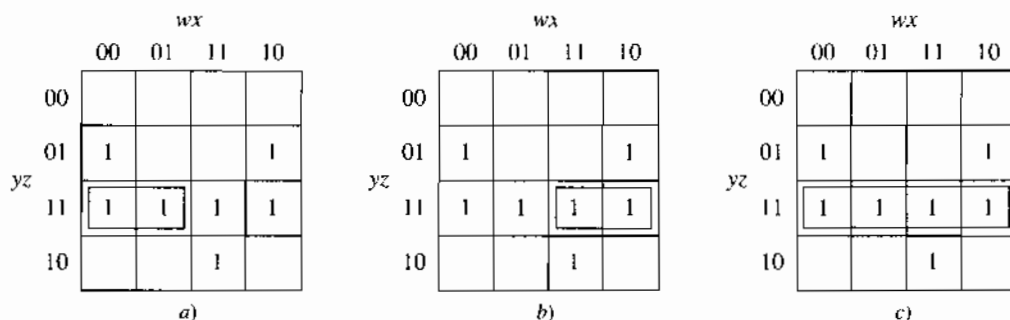


Figura 7. Agrupamientos de minitérminos sobre un mapa.

Un mapa de seis variables consistiría en dos mapas de cinco variables ubicados uno abajo del otro. En la parte superior, la primera variable tomaría un valor, digamos 0, y la parte inferior representaría el valor 1. (Dibuje un mapa de este tipo y manténgalo a la mano para uso posterior.) Este esquema se puede repetir conceptualmente para mapas de orden superior. Pero puesto que es difícil visualizar las interrelaciones de las celdas, la utilidad práctica se reduce claramente para más de seis variables.

Cubos de orden k

La idea que acaba de explicarse tiene algunas consecuencias de largo alcance, las cuales se considerarán en seguida. El tema se iniciará a partir de la función descrita en el ejemplo 1 y especificada por el mapa de la figura 5b, que se repite en la figura 7a. Los minitérminos m_3 y m_7 son lógicamente adyacentes. La suma de estos dos minitérminos es:

$$w'x'yz + w'xyz = w'yz(x + x') = w'yz$$

Las tres literales comunes en los minitérminos pueden factorizarse mediante la ley distributiva, como se muestra, dejando un factor $x + x' = 1$ por la definición del complemento.

De este modo, la suma de los dos minitérminos se simplifica a un solo término consistente en tres literales comunes. Este agrupamiento de los dos minitérminos se muestra encerrado dentro de un rectángulo en la figura 7a.⁸

Es posible efectuar un agrupamiento similar de los minitérminos m_{11} y m_{15} para producir $wx'yz + wxyz = wyz$. Esto se muestra en el mapa en la figura 7b; al encerrar los dos minitérminos. Los resultados de los dos agrupamientos precedentes de minitérminos, $w'yz$ y wyz tienen la propiedad de que todas las literales son comunes excepto una; la diferente aparece complementada en un término y no en el otro. El patrón debe ser evidente. La adición de los dos términos debe producir también la eliminación de esta literal impar:

$$w'yz + wyz = yz$$

El resultado es que un agrupamiento de cuatro minitérminos se ha reducido a un término con sólo dos literales. Este agrupamiento de cuatro minitérminos en un renglón se muestra encerrado en la figura 7c.

⁸ En otros libros podría encerrarse mediante un círculo o una elipse. Para simplificar, diremos "encerrar", entendiendo que es "encerrar en un rectángulo."

		wx			
		00	01	11	10
yz	00				
	01	1			1
	11	1	1	1	1
	10			1	

Parte de la respuesta del ejercicio 5.

Ejercicio 5

- El ejemplo anterior mostró cuatro minitérminos que eran adyacentes en dos pares, los cuales se combinaron después en un grupo rectangular de 2^2 celdas o cubo 2. Existen otros dos pares de estos mismos cuatro minitérminos que producen el mismo cubo 2. Encuentre estos dos pares.
- El mapa en la figura 7 contiene otros cuatro minitérminos, además de los que acaban de describirse, que son adyacentes en pares, y que forman cuatro diferentes cubos 3. Pares apropiados de estos cubos 3 forma en conjunto un cubo 2. Encierre dos pares apropiados (en rectángulo) y escriba una expresión simplificada para su suma. Repita con los otros dos pares y confirme que los resultados finales son los mismos.
- Hay otro par de minitérminos adyacentes. Enciérrelos sobre el mapa y escriba una expresión simplificada para la suma de los dos minitérminos.

Respuesta⁹

Es posible generalizar lo que se acaba de presentar mediante un ejemplo. Una función de n variables tendrá el valor 1 en algunas celdas de su mapa lógico y el valor 0 en otras. Para distinguir entre estas celdas, vamos a referirnos a ellas como las celdas de 1s y las celdas de 0s, respectivamente. Con base en el ejemplo anterior, formulamos la siguiente definición:

Un conjunto de 2^k celdas de 1s, cada una de las cuales es adyacente a otras k en el conjunto, se denomina un cubo de orden k , o un cubo k en forma abreviada. Se dice que el cubo k cubre cada una de las 2^k celdas.

Volvamos a la figura 7c y apliquemos este nuevo lenguaje. El conjunto {14, 15} es un cubo 1 (cuenta con $2 = 2^1$ celdas, por lo que $k = 1$). Los conjuntos {3, 7, 11, 15} y {1, 3, 9, 11} son cubos 2. (Verifique la afirmación.) Advierta que dentro del cubo 2 {3, 7, 11, 15} hay cuatro cubos 1: {3, 11}, {3, 7}, {7, 15}, y {11, 15}. Sin embargo, puesto que estos cubos 1 son cubiertos completamente por el cubo 2 más grande, la última expresión no necesita incluirlos.

Ejercicio 6. Escriba la suma de las expresiones de conmutación correspondientes al cubo 2 {3, 7, 11, 15} y los cubos 1 cubiertos por él: {3, 11}, {3, 7}, {7, 15} y {11, 15}. Especifique la ley de conmutación que permite la simplificación de esta expresión, y entonces simplifíquela lo más posible.

Respuesta¹⁰

⁹ a) 0111 y 1111 forman en conjunto xyz ; 0011 y 1011 forman $x'yz$; se combinan en la misma yz . b) $\{m_1, m_3\}$ y $\{m_9, m_{11}\}$; $(w'x'y'z + w'x'yz) + (wx'y'z + wx'yz) = x'z$. La adyacencia de pares se muestra dibujando un arco que une los dos pares en la tabla. Otra posibilidad es unir $\{m_1, m_9\}$ y $\{m_3, m_{11}\}$ y entonces unir los cubos 3. Confirme que este agrupamiento produce el mismo resultado final. c) $\{m_{14}, m_{15}\}$; $wxyz + wxyz' = wxy$. (El mapa se muestra dentro del texto.)

¹⁰ La ley de absorción.

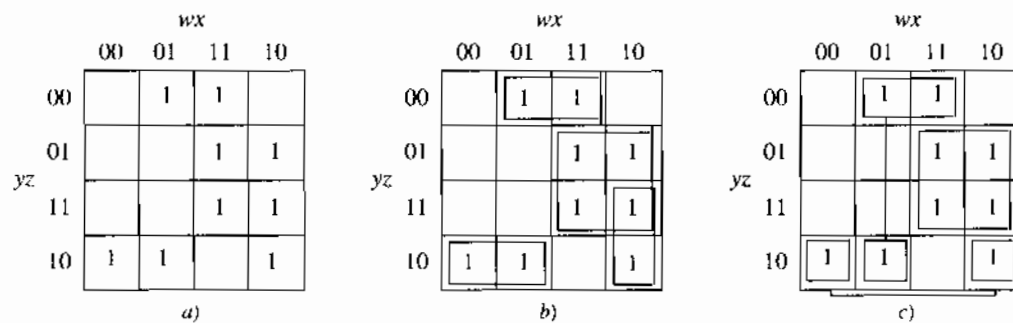


Figura 8. Mapa de $f = \Sigma(2, 4, 6, 9, 10, 11, 12, 13, 15)$ y sus cubos k .

EJEMPLO 2

Una función se especifica mediante la siguiente lista de minitérminos:

$$F = \Sigma(2, 4, 6, 9, 10, 11, 12, 13, 15)$$

- Construya el mapa lógico.
- Liste todos los cubos k posibles.
- Liste todos los cubos k que no son cubiertos por ningún cubo k de orden superior.
- Liste un conjunto mínimo de cubos k que cubra todas las celdas de 1s al menos una vez. Repita para todos los conjuntos mínimos que pueda encontrar.

Respuesta

- El mapa de cuatro variables se muestra en la figura 8.
- Empiece con el número de minitérmino más bajo; construya todos los cubos 1 formados con número minitérmino superiores. Repita con cada número de minitérmino subsecuente hasta que todos se agoten (¡esperando que no sea usted quien se agote!). Construya cubos 2 de la misma manera, y después suprima todos los cubos 1 cubiertos por el último. Los cubos de orden k superiores se construyen de manera similar. El resultado: {2, 6}, {2, 10}, {4, 6}, {4, 12}, {10, 11}, {12, 13}, {9, 11, 13, 15}; más cubos 1 {9, 11}, {9, 13}, {11, 15}, {13, 15}, la totalidad de los cuales son cubiertos por el cubo 2. Éste es evidentemente un procedimiento algorítmico para el cual sería posible escribir un programa (que ya ha sido hecho).
- Todos excepto los últimos cuatro en b. Siete términos serían necesarios en una expresión de suma de productos si todos se usaran, pero no todos serían necesarios.
- Además del cubo 2, que cubre todos los minitérminos impares, se necesitan los cubos 1 para cubrir los cinco minitérminos restantes, todos pares: 2, 4, 6, 10, 12. Dos cubos 1 solos no pueden cubrir la totalidad de los cinco minitérminos, de modo que se necesitan al menos tres cubos 1. Dos de estos tres deben cubrir cada dos distintos minitérminos pares. Además del cubo 2, hay cuatro combinaciones diferentes de tres cubos 1 que efectúan el trabajo. Dos de estos conjuntos de k cubos que cubren todos los minitérminos se muestran encerrados en las partes a y c de la figura. (Encuentre usted los otros dos.) Cada una de estas cuatro posibilidades tendrá el mismo número de términos y el mismo número de literales.¹¹ Confirme todo esto y escriba las cuatro expresiones. ♦

¹¹ Piense en términos de la implementación eventual en compuertas discretas, si cada minitérmino cuesta lo mismo, cada operación AND tiene el mismo costo y ocurre lo mismo con cada OR, podríamos afirmar que cada una de estas implementaciones costará lo mismo. Sin embargo, unos cuantos circuitos digitales, si es que hay alguno, se implementan en compuertas discretas con tecnología actual. Más adelante se ampliará este asunto.

3 REALIZACIONES MÍNIMAS DE FUNCIONES DE CONMUTACIÓN

El ejemplo anterior muestra un caso en el que, con el uso de un mapa lógico, pueden obtenerse cuatro expresiones de conmutación diferentes que representen la misma función, donde cada una de las expresiones tiene el mismo número de términos, y el mismo número de literales por término. Es imposible reducir aún más estas expresiones, ya sea omitiendo un término o eliminando una literal de un término, sin cambiar el valor lógico de la función.

Expresiones irreducibles y mínimas

Con base en el párrafo anterior, establecemos la siguiente definición:

Una expresión de suma de productos es irreducible sin ningún término, o ninguna literal de cualquier término; puede eliminarse sin cambiar el valor lógico de la expresión.

Por consiguiente, las expresiones que se obtuvieron en el ejemplo anterior resultan irreducibles.

Surge otra idea cuando comparamos diversas expresiones que representan una función. La definimos del siguiente modo:

Una expresión de suma de productos equivalente a una función es mínima si ésta tiene el menor número de términos que cualquier otra expresión equivalente a dicha función; si hay más de una expresión que tenga el menor número de términos, aquella con menos literales es mínima.

En el ejemplo 2, la totalidad de las cuatro expresiones tiene el mismo número de términos y el mismo número de literales por término.

Ninguna otra expresión equivalente tiene menos términos (o menor número de literales por término); por consiguiente, todas ellas son mínimas. Esto demuestra que la existencia de una expresión mínima que representa a una función no implica necesariamente que ésta es única. Existe otra expresión, equivalente a aquéllas consideradas en el ejemplo (entre otras, la suma de todos los cubos k). Sin embargo, ninguna otra es mínima. ¿Es posible que una expresión sea mínima pero no irreducible?

Consideraremos cuidadosamente las dos definiciones. Suponga que una expresión es mínima pero que puede reducirse al eliminar un término o una literal en un término. Esto es una contradicción, puesto que la capacidad para reducirla significa que no es mínima. La conclusión es que si una expresión es mínima, resulta necesariamente irreducible.

¿Qué sucede con la otra alternativa? ¿Es posible que una expresión sea irreducible pero no mínima? La definición de una expresión irreducible no dice absolutamente nada respecto al número de términos en esa expresión. Por consiguiente, no hay ninguna razón por la cual una expresión irreducible *deba ser* mínima; ¡puede o no serlo!

Ejercicio 7. Regrese al ejemplo 2 y escriba la expresión que incluye el cubo 2 y los siguientes cubos 1: {2, 6}, {4, 6}, {10, 11}, y {12, 13}. Presente argumentos que demuestren que esta expresión es irreducible pero no mínima.

Respuesta¹²

¹² La expresión que contiene los cinco términos correspondientes no es mínima, ya que la expresión mínima que se determinó antes tenía cuatro términos. El que la expresión sea irreducible, sin embargo, se establece sin dificultades advirtiendo que cada uno de los cinco cubos incluye un minitérmino que no comprende ningún otro cubo. (Asegúrese de confirmar lo anterior.)

Implicantes primos

Varios conceptos nuevos se presentaron en la sección anterior: expresiones *mínima e irreducible*, y la *cobertura* de un cubo k por otro. Estos conceptos se formalizarán ahora y se extenderán a las funciones de conmutación en general. El concepto de cobertura, por ejemplo, puede extenderse de la manera siguiente:

La función de conmutación f_1 cubre a la función de conmutación f_2 si, siempre que $f_2 = 1$, $f_1 = 1$.

Supóngase, como ejemplo, que una función $f = xy'z + wyz' + wx'y + xz'$ y que una función g es igual a los últimos dos términos: $g = wx'y + xz'$. Siempre que $g = 1$, entonces $f = xy'z + wyz' + 1 = 1$. Por consiguiente, f cubre a g .

En una expresión de suma de productos, es evidente a partir de esta ilustración que siempre que cualquiera de los términos es 1, la propia expresión es 1. Esto quiere decir que una suma de productos cubre todo término producto en la expresión. En este caso especial de una función que cubre un producto de literales, formulamos la siguiente definición:

Si una función f cubre un producto de literales, entonces el producto de literales implica a f o es una implicante de f .

En el caso inmediatamente precedente por ejemplo, el primer término, $xy'z$, es un producto de literales. Siempre que $xy'z = 1$ la función completa f se convierte en 1. De tal manera, f cubre $xy'z$ y, en la nueva terminología, $xy'z$ es una implicante de f . Ahora bien, un producto de literales puede ser un minitérmino o representar un cubo k que cubre a 2^k minitérminos. *El mismo lenguaje de cobertura se aplica a cubos k como una función en general.* Puesto que un cubo k cubre cada una de las 2^k celdas de 1s que conforman el cubo k , cada celda es un implicante del cubo k .

Como ejemplo, regrese al ejemplo 2 en la sección anterior. Ahí el cubo 2 {9, 11, 13, 15} cubre a cada uno de los cubos 1 {9, 11}, {9, 13}, {11, 15}, {13, 15}. Cada cubo 1, a su vez, cubre a cada uno de sus dos minitérminos. Una expresión que representa a la función es:

$$f = wz + xy'z' + w'yz' + wx'y + wxyz$$

Cada uno de los términos producto a la derecha es un implicante de f . Pero uno de éstos (con dos literales) representa un cubo 2, cada uno de esos términos con tres literales representa un cubo 1, y el último es un minitérmino. Suponga que la literal x se elimina de $wxyz$, dejando wyz . De acuerdo con el mapa en la figura 8, esto corresponde a un cubo 1 cubierto por el cubo 2 {9, 11, 13, 15} representado por wz . Tanto el cubo 1 como el cubo 2 son implicantes de f . Debe de haber alguna manera de distinguir diferencias de este tipo entre implicantes. La distinción se plantea en la siguiente definición:

Una implicante de una función f es un implicante primo p si la omisión de cualquier literal de p produce un producto de literales que no es un implicante de f .

Así, en el ejemplo anterior, $wxyz$ es un implicante, aunque no es un implicante primo, debido a que al eliminar una de las literales x o y , o ambas, originarán un implicante de f .

Existe una distinción entre implicantes y cubos k , aunque también hay una relación. Una implicante (incluyendo un implicante primo) es un producto de literales; un cubo k , en cambio, corresponde a un conjunto de 2^k celdas de 1s en un mapa, cada una de las cuales es adyacente a las otras k . Cada celda de 1s corresponde a un minitérmino; la suma de los 2^k minitérminos correspondientes a un cubo k constituye el implicante. Si un cubo k no es cubierto por un cubo de orden superior, el implicante correspondiente es un implicante primo.

De ese modo, para determinar todos los implicantes primos de una función, ubicamos en el mapa lógico de la función todos aquellos cubos k que no son cubiertos por cubos de orden superior.

Expresio

		wx				Implicantes	Cubos k
		00	01	11	10		
yz	00		1	1		{9, 11, 13, 15}	wz
	01			1	1	{2, 6}	w'yz'
	11			1	1	{2, 10}	x'yz'
	10	1	1		1	{4, 6}	w'xz'
						{4, 12}	xy'z'
						{10, 11}	wx'y
						{12, 13}	wxy'

Figura 9. Función de ejemplo con cubos k e implicantes correspondientes.

El valor del concepto de implicante primo surge del siguiente teorema:

Si una expresión de suma de productos que representan una función de conmutación f es irreducible, entonces es una suma de implicantes primos.

Ejercicio 8. Demuestre el teorema anterior por contradicción. Esto es, suponga que un producto es un implicante (cada término en una suma *debe ser* un implicante) pero no un implicante primo, y llegue a una contradicción. ♦

Expresiones mínimas de suma de productos¹³

Ahora es claro el proceso para determinar una expresión mínima de suma de productos que es equivalente a una función de conmutación f :

1. Encuentre todos los implicantes primos.
2. Elija el subconjunto más pequeño posible de implicantes primos, asegurando que se cubren todos los minitérminos.

La única tarea que resta es ilustrar el proceso con ejemplos.

EJEMPLO 3

El mapa de la función que se muestra en la figura 8 se repite en la figura 9, junto con una lista de todos los cubos k no cubiertos por algún cubo de orden superior, y sus implicantes correspondientes. El ejemplo se utilizará para ilustrar cómo determinar una expresión mínima de suma de productos que represente una función determinada.

Cada cubo k listado en la figura corresponde a un implicante primo. (Verifique esta afirmación.) Puesto que la función tiene cuatro variables, los cubos 2 corresponden a un implicante primo de $4 - 2 = 2$ literales, y todos los cubos 1 corresponden a un implicante primo que cuenta con $4 - 1 = 3$ literales. El examen de los cubos k listados indica que todas las celdas de 1s aparecen en más de un cubo salvo las celdas 9 y 15, que sólo aparecen en uno. (El ejemplo se continuará más adelante.) ■

Si bien surge de un ejemplo, esta distinción constituye la base de una definición general:

¹³ En tiempos anteriores, el interés por encontrar expresiones mínimas que representaran a una función se relacionaba con la reducción del costo al disminuir el número de compuertas primitivas o el número de entradas por compuerta. Aunque esta motivación es menos demandante en la actualidad, sigue siendo importante, en especial en los dispositivos llamados *lógicos programables*, que se presentarán en los capítulos 4 y 8.

		x			
		0	1		
yz	00	1	1		
	01		1	{0, 4}	$y'z'$
	11	1	1	{4, 5}	xy'
	10	1		{5, 7}	xz
				{7, 3}	yz
				{3, 2}	$x'y$
				{2, 0}	$x'z'$

Figura 10. Mapa cíclico.

Un implicante primo es un implicante primo esencial si cubre al menos un minitérmino no cubierto por cualquier otro implicante primo. Cualquier minitérmino que es cubierto por únicamente un implicante primo recibe el nombre de minitérmino distinguible.

La importancia de un implicante primo esencial es que *debe* incluirse en cualquier expresión mínima que represente una función; en otro caso el minitérmino que distingue no se cubrirá.

En consecuencia, el proceso de buscar una expresión mínima, procede de la manera siguiente:

1. Se identifican todos los implicantes primos esenciales. Si este conjunto cubre todos los minitérminos, finaliza la tarea.
2. Se busca el número menor de implicantes primos no esenciales que cubren los minitérminos no cubiertos por los implicantes primos esenciales.
3. Si se lleva a cabo una elección en el paso 2, elíjanse los implicantes primos no esenciales con el menor número de literales.

EJEMPLO 3 (continuación)

Regresemos ahora al ejemplo. El implicante primo wz que corresponde al cubo 2 es esencial. Cubre cuatro minitérminos (dejando que sólo cinco sean cubiertos por otros implicantes primos), cada uno de los cuales cubre dos minitérminos. Por consiguiente, serán necesarios al menos tres implicantes primos más (dos más cubrirían únicamente cuatro minitérminos), dando una expresión mínima de cuatro implicantes primos. En el ejemplo 2, utilizando otro procedimiento, encontramos cuatro expresiones mínimas que contienen cuatro de los que podemos ahora identificar como implicantes primos. ■

EJEMPLO 4

La siguiente función tiene una estructura interesante de implicantes primos: $f(A, B, C) = \Sigma(0, 2, 3, 4, 5, 7)$. El mapa, junto con la lista de cubos 1 e implicantes primos, se muestra en la figura 10.

Hay seis cubos 1: {0, 4}, {4, 5}, {5, 7}, {7, 3}, {3, 2}, y {2, 0}, representando todos implicantes primos. En forma colectiva, éstos tienen cierto patrón, subrayado por el orden en el que se escriben los cubos 1 y sus minitérminos constituyentes. Se afirma que un mapa de estas características será *cíclico*. Cada minitérmino aparece en exactamente dos implicantes primos. Así, es posible cubrir todos los minitérminos por medio de dos conjuntos de tres implicantes primos cada uno, sin minitérminos comunes:

$$f = y'z' + xz + x'y = x'z' + xy' + yz$$

(Encierre los minitérminos correspondientes y advierta el patrón.) ■

Ejercicio 9. Construya el mapa y encuentre una expresión mínima de suma de productos equivalente a la siguiente función:

$$f(A, B, C, D) = \Sigma(0, 2, 3, 4, 8, 9, 10, 14)$$

Comente cualquier aspecto inesperado que observe.

Respuesta¹⁴

Expresiones mínimas de producto de sumas

Debido al principio de dualidad, todo lo que se ha hecho en términos de expresiones de suma de productos puede repetirse para llegar a expresiones de productos de sumas. Advierta que la idea de adyacencia se aplica a las celdas de 0s y a las celdas de 1s. Son obvios los cambios necesarios en las definiciones de conceptos tales como cubos k , cobertura, expresiones irreducibles e implicantes primos, con cambios evidentes en la terminología. Así,

Un conjunto de 2^k celdas de 0s, cada una de las cuales es adyacente a otras k en el conjunto, se denomina cubo k ; éste cubre cada una de las 2^k celdas de 0s.

Si una función cubre una suma de literales, entonces la suma de literales implica a f o es una implicante de f . El implicante es un implicante primo si la eliminación de cualquier literal origina una suma de literales que no es un implicante de f .

*Si una expresión de producto de sumas equivalente a una función f es irreducible, entonces ésta debe ser un producto de implicantes primos. Un implicante primo es esencial si cubre al menos un maxitérmino no cubierto por otros implicantes primos. Un maxitérmino que es cubierto por únicamente un implicante primo es distinguible.*¹⁵

A partir de todo lo anterior, se concluye que una expresión mínima de producto de sumas debe contener a todo implicante primo esencial. Podría contener también otros implicantes primos que no son esenciales.

El proceso de determinar una expresión mínima p de s a partir de una lista de maxitérminos puede diferir un poco, en detalles, del proceso relativo a la determinación de una expresión mínima de p a partir de una lista de minitérminos, aunque todos los pasos son duales.

EJEMPLO 5

Para la función cuya lista de minitérminos se dio en el ejemplo 2 y cuyo mapa lógico se muestra en la figura 8a.

- Escriba la expresión que representa la función como una lista de maxitérminos.
- Especifique un método para obtener una lista de todos los cubos k que no son cubiertos por ningún cubo k de orden superior; después de eso obtenga la lista.
- Escriba los factores suma para cada implicante primo.
- Escriba una expresión mínima de productos de sumas para la función. ¿Hay más de una?

¹⁴ Hay cinco implicantes primos, uno correspondiente a un cubo 2 y cuatro correspondientes a cubos 1: $\{0, 2, 8, 10\}$, $\{0, 4\}$, $\{2, 3\}$, $\{8, 9\}$, $\{10, 14\}$. Los cuatro cubos 1 son esenciales, de modo que *deben* incluirse en la expresión mínima. En conjunto cubren todos los minitérminos, por lo que no se necesita el implicante primo restante, aun cuando éste cuente con menor número de literales. La expresión mínima es $f = A'C'D' + A'B'C + AB'C + ACD'$. *

¹⁵ Para distinguir entre los casos s de p y p de s , algunos autores utilizan *implicado* e *implicado primo* para el caso p de s en lugar de *implicante* e *implicante primo*. Usaremos el mismo término en ambos casos puesto que es mínima la probabilidad de confusión.

Respuesta

- a. Todos los cuadrados sobre el mapa que no son 1 deben ser 0. Por consiguiente, la lista es:

$$\prod (0, 1, 3, 5, 7, 8, 14) = M_0 M_1 M_3 M_5 M_7 M_8 M_{14}$$

- b. Un método consiste en encerrar de manera apropiada los ceros sobre el mapa; el resultado es $\{1, 3, 5, 7\}$, $\{0, 8\}$, $\{14\}$, $\{0, 1\}$.
- c. Los factores suma son $(w + z')$, $(x + y + z)$, $(w' + x' + y' + z)$, y $(w' + x' + y')$.
- d. Los primeros tres factores en c son implicantes primos esenciales, y en conjunto cubren a todos los maxitérminos. Por consiguiente, sólo habrá una expresión de este tipo:

$$f = (w + z')(x + y + z)(w' + x' + y' + z)$$

Implementaciones de dos niveles

La formulación precedente en este capítulo se ha concentrado en la obtención de diferentes expresiones para representar una función de conmutación determinada. Debe ser posible *implementar* cualquiera de estas expresiones utilizando compuertas lógicas primitivas, y eso es lo que nos disponemos a llevar a cabo.

Implementación AND-OR

Una expresión que puede representar una función de conmutación es una suma de productos de variables de conmutación. Un producto lógico tal como xyz' , por ejemplo, es la AND de tres variables: x , y y z' . Por tanto, puede implementarse por medio de una compuerta AND con tres entradas. Cada término en una expresión s de p presenta exactamente la misma forma, difiriendo quizá sólo en el número de variables. Por tanto, cualquier término puede implementarse mediante una compuerta AND, posiblemente con un número diferente de entradas en cada caso.

La expresión s de p completa es la suma lógica de tales términos producto. Puesto que la suma lógica se implementa por medio de una compuerta OR, entonces la expresión completa se implementa mediante una compuerta OR cuyas entradas son las salidas de las compuertas AND que implementan cada producto lógico.

Vamos a usar como ejemplo la expresión mínima que se obtuvo en el ejercicio 9. Consiste en la suma lógica de cuatro términos, cada uno de los cuales es el producto lógico de tres variables. Por consiguiente, la expresión puede efectuarse mediante cuatro compuertas AND de tres entradas cuyas salidas (cuatro en total) son entradas para una compuerta OR. Las variables complementadas se obtienen mediante inversores. El resultado se muestra en la figura 11.

Algunas de las variables en la expresión se complementan y otras no. Como ya se mencionó en la sección 7 del capítulo 2, con mucha frecuencia se cuenta tanto con la variable de con-

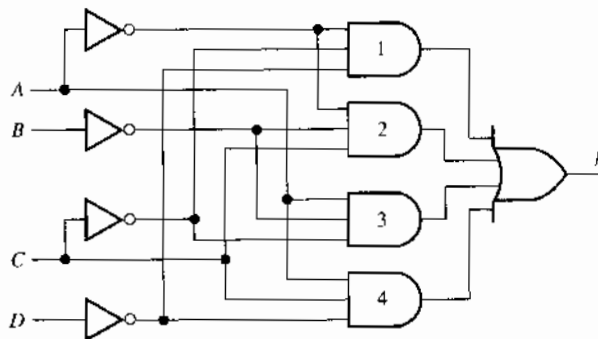


Figura 11. Implementación de dos niveles de la función del ejercicio 9.

mutación como con su complemento.¹⁶ En consecuencia, no se necesita efectuar nada especial para obtener el complemento, aunque en la figura 11 se supone que sólo la variable se dispone como una entrada. (Suponga que tanto las variables como sus complementos están disponibles. Muéstrese la implementación resultante de la función).

Sin contar los inversores, que de igual modo tal vez no estén presentes, la figura 11 es un circuito de dos niveles: todas las señales externas deben atravesar las dos compuertas desde la entrada inicial hasta la salida. En realizaciones TTL estos circuitos proporcionan menos retraso que cualquier otro circuito que podría efectuar la misma función.

Desde luego, la expresión *canónica* de suma de productos escrita a partir del listado de minterminos tiene exactamente la misma forma que la suma de productos *mínima* que se encontró en el ejercicio 9.

Por tanto, un circuito que la implemente será también uno de dos niveles. (Especifique el número de compuertas que esta implementación tendrá y el factor de carga de salida de cada compuerta para el caso anterior. Compare. Suponga que se disponen tanto de las variables como de sus complementos, y bosqueje el circuito.)

En ambas de las anteriores implementaciones de la misma función, los factores de carga de entrada de todas las compuertas AND son las mismas. Esto no es necesariamente cierto para todas las expresiones s de p .

Ejercicio 10

- Utilizando el mapa de la figura 8b, escriba la expresión mínima s de p que resulta de los cubos k encerrados.
- Dibuje un circuito AND-OR de dos niveles que implemente esta expresión y compare los factores de carga de entrada de las compuertas AND.

Implementación NAND

Las compuertas primitivas del tipo utilizado en los ejemplos precedentes se implementan universalmente con circuitos integrados a pequeña escala, como se explicó en el capítulo 2.

Los circuitos SSI específicos por lo común incluyen varias copias del mismo tipo de compuerta, teniendo todas el mismo factor de carga de entrada. (Regrese a la figura 23 en el capítulo 2.) Al usar circuitos de este tipo, las implementaciones AND-OR de dos niveles requieren el uso de dos tipos SSI diferentes, aun cuando todas las compuertas AND tienen el mismo factor de carga de entrada. La implementación física sería más simple si únicamente un tipo de compuerta se utilizara en las implementaciones lógicas. La solución a este problema se logra a partir de dos observaciones. Una surge de la figura 9 en el capítulo 2, la cual muestra dos formas equivalentes de una compuerta NAND (consúltela), y la segunda proviene al advertir que dos inversiones consecutivas de una señal producen esa misma señal.

En el circuito de la figura 11, suponga que la totalidad de las compuertas AND se sustituyen por compuertas NAND. El resultado sería inaceptable puesto que la salida de cada compuerta AND se ha invertido. Sin embargo, no hay problema: si se introdujera otra inversión en cada entrada a la compuerta OR, las dos inversiones consecutivas en cada línea desde la salida de la compuerta AND hasta la entrada de la compuerta OR no provocarían cambio en la salida.

Además, la compuerta OR con entradas invertidas es funcionalmente equivalente a una compuerta NAND. Por consiguiente, las compuertas NAND pueden sustituir a todas las compuertas en un circuito AND-OR. Efectúe los pasos y dibuje el circuito NAND resultante.) En este ejemplo particular, la motivación de utilizar únicamente un tipo de pastilla SSI no se ha conseguido por completo; todas las compuertas son NAND, lo que está bien, pero la compuerta de salida no tiene el mismo factor de carga de entrada que las otras.

¹⁶ En el capítulo 5 se mostrará cómo sucede esto.

Implementación OR-AND

Otra forma en la cual puede expresarse una función de conmutación es la forma de producto de sumas. Para implementar una expresión de este tipo, cada suma lógica se realiza por medio de una compuerta OR, y el producto lógico final se implementa por medio de una compuerta AND.

Las compuertas OR pueden o no tener el mismo factor de carga de entrada. Como ejemplo, recurriremos a la expresión p de s que se encontró en el ejemplo 5, repetido aquí:

$$f = (w + z')(x + y + z)(w' + x' + y' + z)$$

Éste es el producto lógico de tres factores, cada uno de los cuales constituye una entrada a una compuerta AND. Estos factores son ellos mismos las salidas de las compuertas OR cuyas entradas corresponden a las literales dentro de los paréntesis. Supondremos que se disponen tanto las variables como sus complementos. La implementación resultante en compuertas primitivas se muestra en la figura 12.

Compare la estructura de este circuito con la de la figura 11. Ambas son realizaciones de dos niveles y, en consecuencia, tienen aproximadamente el mismo retardo. En el caso presente, las tres compuertas OR no tienen el mismo factor de carga de entrada; en realidad, todas son diferentes.

También es cierto, del mismo modo que lo fue en el caso s de p , que la implementación de una expresión canónica p de s es un circuito de dos niveles. Suponiendo una forma canónica p de s para la función precedente, ¿cuántas compuertas de primer nivel habrá en una implementación OR-AND, y cuál será su factor de carga de entrada? ¿Cuál será el factor de carga de entrada de la compuerta AND de segundo nivel?

Ejercicio 11. Recorra al hecho de que dos inversiones consecutivas de una variable producen la misma variable, junto con la figura 9 en el capítulo 2, para convertir la figura 12 en un circuito todo NOR. Generalice su resultado a todas las realizaciones OR-AND de dos niveles. ♦

4 IMPLEMENTACIÓN DE EXPRESIONES LÓGICAS

Las primeras partes de este capítulo han presentado formas diferentes de *representar* una función de conmutación, algunas de las cuales parecen ser “más simples” de algún modo. Las únicas implementaciones de funciones lógicas explicadas hasta ahora fueron los circuitos AND-OR y OR-AND de dos niveles. Puesto que pueden escribirse varias expresiones diferentes para una función de conmutación dada, es probable obtener diferentes realizaciones. Algunas de éstas quizá tengan más rasgos deseables que otras. Aquí consideraremos aspectos de este tipo.

Un factor en el costo de una implementación es el número de compuertas. Sin embargo, en un circuito integrado, no sólo las compuertas toman parte del área del CI, sino que también lo hacen las conexiones entre la salida de compuerta y la entrada a la compuerta del nivel siguiente. Además, en la implementación de dos niveles, la compuerta de salida tiene tantas terminales de entrada como el número de compuertas de primer nivel. Este número, el factor de carga de entrada, puede ser elevado. En algunas tecnologías de CI, el desempeño de una compuerta con

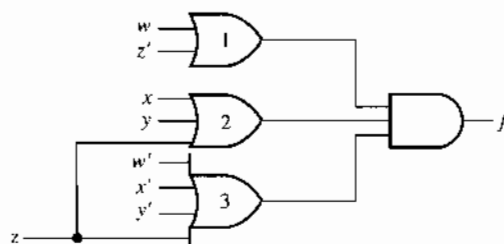


Figura 12. Implementación de producto de sumas.

factor de carga de entrada elevado se degrada de modo correspondiente. Por tanto, algunas veces resulta apropiado un aumento en el número de niveles.

Ejercicio 12. Las siguientes seis expresiones diferentes se pueden escribir para cierta función de conmutación.

$$f(w, x, y, z) = (y + z)(wx + w'x') \quad (a)$$

$$= y(wx + w'x') + z(wx + w'x') \quad (b)$$

$$= wx(y + z) + w'x'(y + z) \quad (c)$$

$$= wxy + w'x'y + wxz + w'x'z \quad (d)$$

$$= m_1 + m_2 + m_3 + m_{13} + m_{14} + m_{15} \quad (e)$$

$$= (y + z)(w' + x)(w + x') \quad (f)$$

- Confirme que cada una de estas expresiones representa la misma función.
- Suponga que cada expresión puede realizarse mediante compuertas AND, OR y NOT. Suponga también que ambas variables y sus complementos se encuentran disponibles a partir de una fuente externa.¹⁷ Encuentre una realización de cada expresión.
- Suponga que todas las compuertas tienen el mismo retardo t_p . Construya una tabla cuyos renglones correspondan a las diferentes implementaciones y cuyas columnas proporcionen el número de compuerta, el número de conexiones internas, el retardo mayor de la entrada a la salida, y el factor de carga de entrada más grande de cualquier compuerta.

Cada una de las expresiones dadas aquí consisten en las operaciones AND, OR y NOT únicamente. Por consiguiente, no debe tener dificultad al determinar una realización para cada expresión. Efectúe usted mismo la implementación de cada uno, antes de confirmarla refiriéndose a las que se dan en la figura 13. A pesar de que sólo hemos efectuado anteriormente implementaciones de dos niveles de funciones de conmutación, hemos visto cómo una compuerta representa cada operación de conmutación. La primera expresión, por ejemplo, es la AND de dos expresiones. De tal manera, la salida proviene de una compuerta AND de dos entradas. Cada una de las dos entradas, a su vez, corresponde a la salida de una compuerta OR de dos entradas. Así, la implementación se efectúa hacia atrás a partir de la salida. ¡Puede usted continuar! ♦

Al principio de este capítulo dedicamos una cantidad de tiempo considerable en la obtención de expresiones canónicas o mínimas $\sum p$ o $\prod p$ de s para una función determinada. Éstas se implementan en circuitos de dos niveles. Como indica el ejercicio anterior, es posible obtener implementaciones multiniveles que tienen menor número de conexiones internas y menores factores de carga de entrada. Como indican las expresiones en los diagramas del circuito del *a*) al *c*), el asunto es determinar factores apropiados de la expresión lógica.

Ejercicio 13

- Para practicar la factorización, considere la expresión en la figura 13*b*. Factorice una expresión común de cada uno de los términos para obtener una expresión equivalente.
- Implemente esta expresión; compare el número de compuertas, el número de conexiones internas y el retardo de propagación a través de la trayectoria más larga con aquella que usted construyó en el ejercicio 12. ♦

¹⁷ Recuerde que los dispositivos denominados *flip-flop*, que se estudiarán en el capítulo 5, siempre ponen en la salida tanto a la variable como a su complemento.

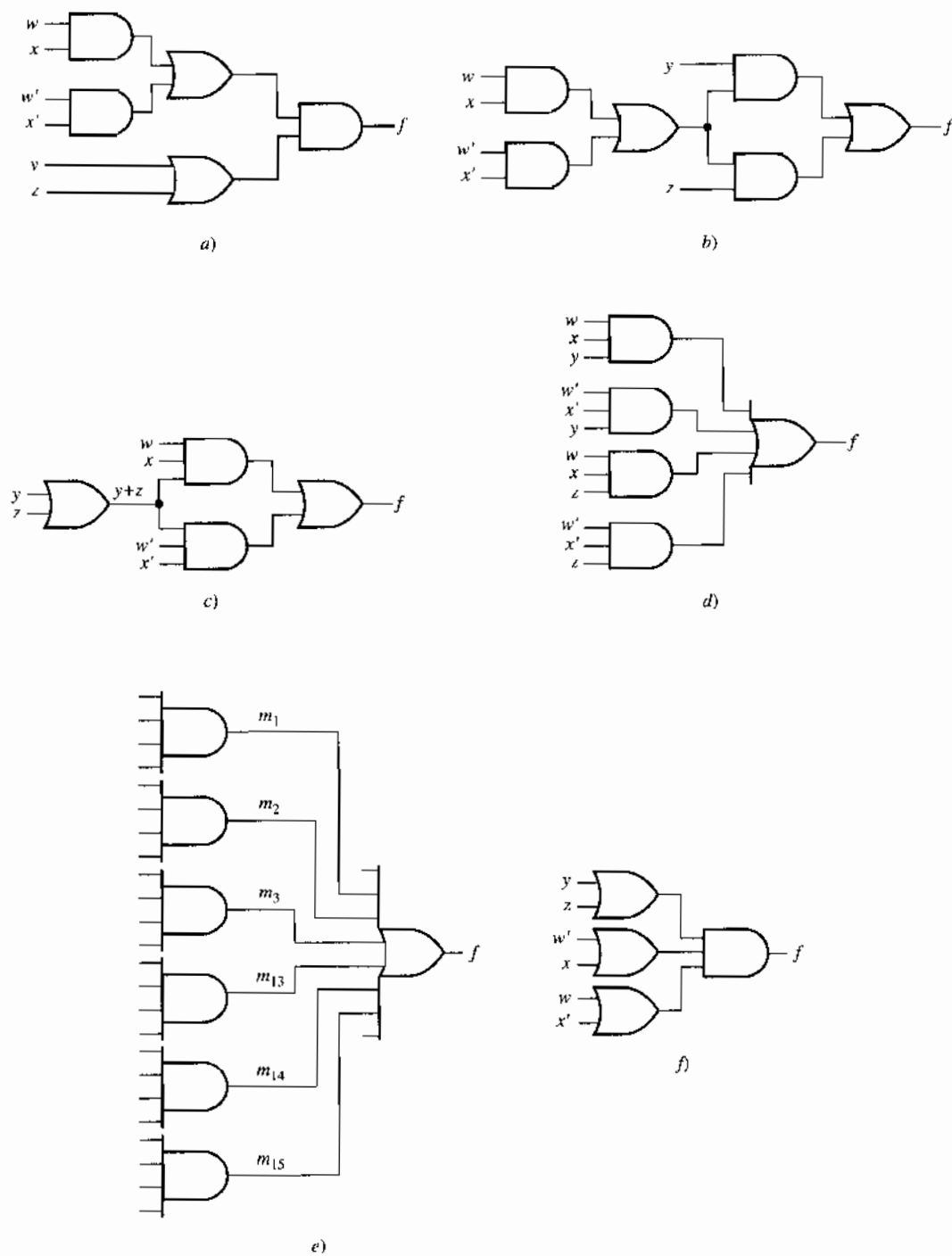


Figura 13. Diferentes circuitos que implementa la misma función.

Análisis

Para una tabla de verdad determinada, lista de minitérminos, mapa lógico o expresión lógica, hemos descrito cómo implementar un circuito que satisfaga la información especificada. ¿Pero cómo saber que no se han cometido errores y que el circuito obtenido realmente tiene las salidas que se especificaron?

En circuitos físicos de todo tipo (no sólo circuitos lógicos), se efectúa un proceso que implica realizar mediciones (de voltaje, digamos) en puntos apropiados en el circuito para verificar que los valores medidos son los que se suponen teóricamente. Este proceso podría denominarse *verificación*.

¿Pero por qué implementar primero físicamente el circuito, antes de la verificación? Una vez que se ha obtenido un circuito lógico en papel (o generado mediante software), es posible *analizar* el circuito para verificar que los valores lógicos en cualquier punto son en realidad los requeridos por las especificaciones de diseño. Comparado con el proceso de diseño, el análisis de circuitos lógicos es bastante simple. Es factible asignar un nombre a cualquier salida de compuerta. Las expresiones lógicas para estas salidas se escriben en términos de las entradas a esas compuertas. Cada entrada de una compuerta es una entrada primaria o la salida de otra compuerta.

Si el proceso se efectúa para las salidas de todas las compuertas, a la larga sólo las entradas primarias quedarán en las expresiones correspondientes a cualquier salida de compuerta, incluso en aquellas a partir de las cuales se toman las salidas del circuito. Estas expresiones se comparan después con información proporcionada. Esto debe ser así y no es algo complicado. En el caso de la implementación de dos niveles en un circuito de una salida, el proceso resulta trivial. En otros casos, y con circuitos que tienen más de una salida, quizá haya que hacer algo adicional.

Considere como ejemplo el circuito en la figura 13b y denominemos u la salida de la compuerta OR a la izquierda. Las entradas a esta compuerta OR son las salidas de las dos compuertas AND a su izquierda, que corresponden, respectivamente, a wx y $w'x'$. Por consiguiente, esta salida de compuerta OR es $u = wx + w'x'$. La salida del circuito es la salida de la compuerta OR a la derecha: $f = yu + zu$. Sustituyendo la expresión para u , esto confirma la expresión a partir de la cual se implementó el circuito.

Características de circuitos de compuerta

En el capítulo anterior se explicaron las siguientes características de los circuitos lógicos: factor de carga de entrada, factor de carga de salida, velocidad (o la propiedad opuesta, retardo de la propagación) y niveles. Las implementaciones en la figura 13 difieren en estos aspectos. Advierta que las implementaciones de suma de productos y de producto de sumas, ya sean canónicas o reducidas, corresponden a circuitos de dos niveles y por ello tienen el menor retardo en la tecnología TTL. Sin embargo, las realizaciones canónicas son las más derrochadoras en términos de número de compuertas.¹⁸ A partir de su estructura, las realizaciones s de p se describen como circuitos AND-OR, en tanto que las p de s corresponden a circuitos OR-AND. Las únicas compuertas en cada una de las implementaciones en la figura 13 son AND y OR.

Como se explicó respecto a la implementación de la figura 11, resultaría de gran valor si sólo se usara un tipo de compuerta en cada implementación; entonces serían necesarios paquetes SSI con únicamente un tipo de compuerta. En el capítulo 2 encontramos que las compuertas NAND son universales, de igual manera que lo son las compuertas NOR. Por consiguiente, la función implementada por los circuitos en la figura 13 debe ser realizable mediante un circuito consistente sólo de NAND o exclusivamente de NOR. Una manera de efectuar lo anterior es convertir cada uno de los circuitos de la figura 13 a formas equivalentes que utilizan sólo NAND

¹⁸ Lo que es un derroche en el número de compuertas quizá no constituya lo más costoso en términos de CIs SSI.

o NOR. La clave para lo anterior es la figura 9 del capítulo 2 (extendida a cualquier número de variables), con el conocimiento adicional de que una variable permanece sin cambio después de dos inversiones.

Tome como ejemplo el circuito *s* de *p* de la figura 13*d*. Coloque una burbuja de inversión a la salida de cada compuerta AND y balancéela con otra burbuja de inversión en las entradas correspondientes a la compuerta OR; esto significa que no hay cambio en las variables sobre cualquier línea. Utilizando las equivalencias indicadas en la figura 9 en el capítulo 2, todas las compuertas se sustituyen por compuertas NAND, sin modificaciones en ninguna variable.

Ejercicio 14. Confirme este resultado aplicando involución y el teorema de De Morgan a la expresión *s* de *p*. ♦

Se requiere alguna ligera modificación a este procedimiento en cada uno de los circuitos multinivel en la figura 13. En la figura 13*c*, por ejemplo, no hay compuertas AND que se utilizan con la compuerta OR de entrada a fin de efectuar una inversión doble. Si se coloca una burbuja a cada entrada de la compuerta OR, ésta no puede balancearse por medio de burbujas a la salida de compuertas AND, aunque *es posible* balancearla invirtiendo las variables de entrada correspondientes. De tal manera, todas las compuertas pueden ser sustituidas por compuertas NAND, aunque las entradas *y* y *z* deben complementarse. No es gran problema, ya que también se cuenta con entradas complementadas.

Ejercicio 15. Las figuras 13*a* y 13*f* difieren de las otras al tener una compuerta AND a la salida. Demuestre que es factible obtener un equivalente todo NAND, pero a cierto costo. ¿Cuál es dicho costo?

Respuesta¹⁹

Ejercicio 16. De todos los circuitos que implementan la misma función lógica en la figura 13, el que sobresale en términos de número de compuertas, de niveles, de conexiones y factor de carga de entrada máximo es el circuito *p* de *s* de la figura 13*f*. Después de convertir todos los circuitos a circuitos todo NAND, compare las figuras 13*c* y 13*f* en términos del número de compuertas, niveles, conexiones y factor de carga de entrada máximo. ¿Algún comentario? (Recuerde que están disponibles las variables de entrada y sus complementos.) ♦

5 DIAGRAMAS DE TEMPORIZACIÓN

Las representaciones de circuitos lógicos digitales descritas hasta ahora caracterizan únicamente el comportamiento estático: para entradas determinadas, el sistema tiene salidas específicas en el estado estable, como las indicadas en la tabla de verdad. En operación, las entradas cambian con frecuencia, y las salidas efectúan transiciones de un nivel a otro en respuesta a los cambios de las entradas. Como consecuencia de los inevitables retardos de compuerta, sin embargo, las transiciones de salida se retardan en el tiempo respecto a los cambios de la entrada. Esta sección considera de manera breve los efectos de tales retardos en las salidas de circuito.

Dibujar sus formas de onda como una función del tiempo es una buena manera de representar las transiciones de entrada y salida. Tales dibujos, conocidos como *diagramas de temporización*, constituyen otros medios para describir la operación de sistemas digitales. En la figura 14 se ilustra un diagrama de temporización que muestra el comportamiento de una compuerta AND de dos entradas. En el diagrama de temporización las transiciones entre los niveles lógicos se dibujan comúnmente como líneas verticales, aunque en realidad las líneas tienen una pendiente

¹⁹ El aumento de retardo resulta de una NAND añadida a la salida de acuerdo con (26) en el capítulo 2.

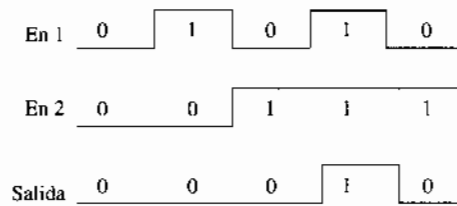


Figura 14. Diagrama de temporización para una compuerta AND.

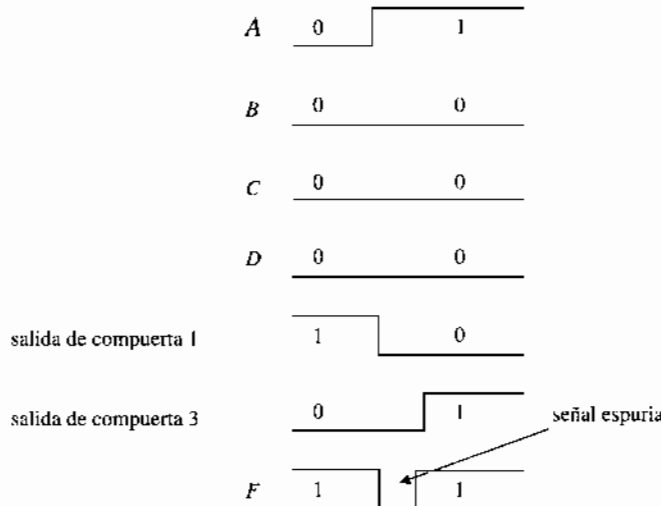


Figura 15. Riesgo de temporización en el circuito de la figura 11.

finita, como se ilustra en la figura 22 del capítulo 2. Si el diagrama de temporización se dibujara con estimaciones de los retardos de compuerta reales, entonces mostraría el comportamiento dinámico del sistema, en oposición al comportamiento estático que comunica la tabla de verdad.

Si las señales se propagan a través de las compuertas sin retardo, las transiciones de valor lógico ocurrirían en forma instantánea. Sin embargo, las señales inevitablemente se retardan al viajar a través de las compuertas, y diferentes compuertas quizá introduzcan diferentes cantidades de retardo. Así, las señales experimentan diferentes cantidades de retardo cuando recorren distintas trayectorias en un circuito. Como consecuencia, mientras el valor lógico eventual de estado estable en cierto punto en un circuito tal vez sea el esperado, es probable que la salida asuma valores erróneos momentáneos antes de llegar a este valor de estado estable.

Este proceso se ilustra haciendo referencia al circuito que se mostró antes en la figura 11. Suponga que el circuito se encuentra en estado estable con entradas $ABCD = 0000$, y considere que la compuerta 3 tiene un retraso mayor que la compuerta 1. Suponga ahora que la entrada A cambia de 0 a 1. La transición de salida f puede dibujarse como se indica en la figura 15. La salida cambia en forma temporal a 0 antes de alcanzar el valor de estado estable apropiado de 1. Esta transición temporal recibe el nombre de señal espuria. Los diagramas de temporización son las únicas representaciones de sistemas digitales que tienen la capacidad de exhibir tales señales espurias. (La escala de tiempo se ha exagerado.)

En circuitos combinatorios, los riesgos son más una molestia que una amenaza. Sin embargo, siguen siendo indeseables por varias razones. La peor consecuencia ocurre si la salida se observa o muestrea en el momento de la presentación de la señal espuria; en ese caso se muestrea un valor erróneo que puede ocasionar una falla del sistema. Las señales espurias quizá provoquen también ruido de conmutación y disipación de potencia que tal vez resulten indeseables si los circuitos sensibles se encuentran físicamente cerca (a lo mejor sobre la misma pastilla). Como veremos después (capítulo 7), hay muchas circunstancias donde las señales espurias no

tienen efecto adverso sobre la operación o el desempeño del sistema. En tales casos no es necesario que nos preocupemos de ellos.

¿Es posible diseñar circuitos sin riesgo? Para evitar el riesgo mostrado en la figura 15, por ejemplo, es posible diseñar el circuito de la figura 11 de manera que la compuerta 2 tenga un retardo mayor que la compuerta 1. Pero esto no se puede garantizar para cada copia del circuito que se fabrica (quizá millones de copias). Debido a la incapacidad para controlar con precisión los procesos en la manufactura de los circuitos integrados, los retardos de compuertas "idénticas" tienen cierta variación estadística. El valor preciso del retardo en una compuerta específica no puede predecirse; todo lo que es posible es especificar valores mínimo y máximo entre los cuales se encuentra el retardo de una compuerta sobre una CI. El diseño de este circuito para garantizar que el retardo de la compuerta 2 es mayor que el de la compuerta 1 tal vez sea por esa causa imposible o podría requerir un aumento sustancial en el retardo del sistema.

El método adecuado para garantizar la ausencia de riesgo implica incluir una compuerta adicional que mantenga la salida en 0 mientras la entrada A cambia de 0 a 1. Si la implementación del circuito incluye una compuerta AND que incremente el término producto $B'C'D'$, entonces independientemente de los retardos relativos de las compuertas 1 y 3, y de las variaciones en sus retardos, la salida no presentará una señal espuria para esta transición de entrada. (Véase el capítulo 7 para una explicación más completa.)

Ejercicio 17. Para una transición de entrada en $ABCD$ de 0000 a 0010, dibuje un diagrama de temporización para el circuito en la figura 11. ¿Hay un riesgo potencial? Identifique un término producto que deba incluirse en el circuito para eliminar el riesgo cuando las entradas cambian de 0000 a 0010.

Respuesta²⁰

Este breve ejemplo de riesgo se utiliza para subrayar la importancia de las representaciones del diagrama de temporización en el diseño de sistemas digitales. Los riesgos se abordan con mayor detalle en el capítulo 7 vinculados con los circuitos secuenciales asíncronos.

6 FUNCIONES INCOMPLETAMENTE ESPECIFICADAS

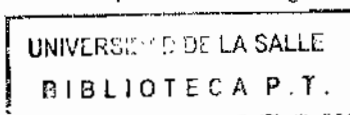
En toda la explicación anterior a este punto, se ha supuesto que, para toda combinación de valores de entrada, cualquier función de conmutación tenga un valor específico definido: ya sea 1 o 0. Una función de este tipo puede llamarse *completamente especificada*.

Valores irrelevantes

Se presentan ocasiones, sin embargo, en las que se sabe que nunca ocurren combinaciones de entrada particulares. En estos casos, ¿qué valores deben asignarse a la *salida*? La respuesta es, *no es relevante*. El valor puede ser 0 o 1, cualquiera que sea más satisfactorio.

Un caso claro incluye un código de 4 bits que representa a los dígitos decimales. (Consulte el capítulo 1 para una explicación de códigos.) Puesto que cuatro variables resultan en 16 combinaciones de valores para representar los 10 dígitos decimales, 6 de las combinaciones posibles no corresponden a los dígitos decimales. Si los 4 bits son entradas a un circuito de conmutación, entonces 6 de las 16 posibles combinaciones de entrada nunca ocurrirán. Por tanto, no tiene importancia lo que la salida podría ser para estas combinaciones de entrada particulares. ¿No resulta razonable, entonces, designar a las salidas con el término de *valores irrelevantes*?

²⁰ El término producto $A'B'D'$ debe incluirse para eliminar el riesgo.



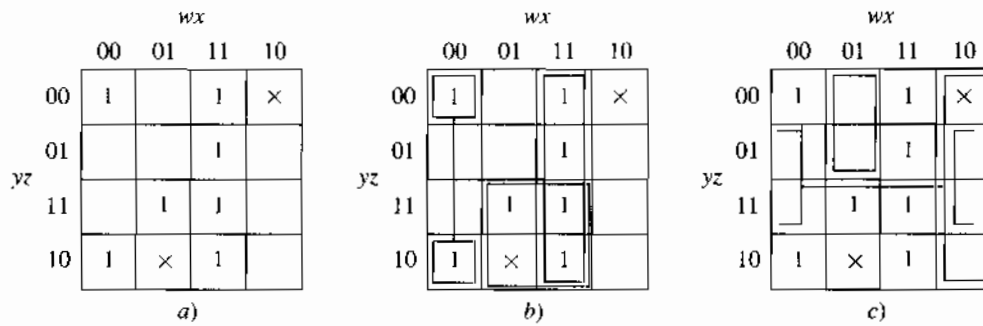


Figura 16. Mapas de $f = \Sigma(0, 2, 7, 12, 13, 14, 15) + \Sigma d(6, 8)$.

Puesto que la salida en estos casos no está completamente especificada, se afirma que la función estará *incompletamente especificada* (¿le sorprende?). Para mostrar una salida irrelevante sobre un mapa, necesitamos un símbolo especial. En este libro recurriremos al “por”, \times .²¹

También necesitamos alguna manera de incluir valores irrelevantes cuando se especifica una función mediante el listado de números de minitérminos. La convención consiste en agregar una lista de números de minitérminos irrelevantes a la lista de minitérminos relevantes, como sigue:

$$f = \Sigma(\text{números de minitérminos}) + \Sigma d(\text{números de irrelevancia})$$

(La d aquí empuja a muchos autores a sumar m antes de la lista de minitérminos. Nosotros no lo haremos.)

EJEMPLO 6

La que sigue es una función incompletamente especificada:

$$f(w, x, y, z) = \Sigma(0, 2, 7, 12, 13, 14, 15) + \Sigma d(6, 8)$$

Nuestra meta es encontrar expresiones mínimas de suma de productos y de productos de sumas que representen esta función. Los mapas con cubos k de 1s encerrados en un caso y cubos k de 0s en los otros se indican en la figura 16.

El cubo 2 {12, 13, 14, 15} en la figura 16b es esencial, distinguible por el minitérmino 13. El minitérmino 7 puede formar un cubo 1 con el minitérmino 15, aunque el implicante primo resultante BDC tendrá tres literales. Sin embargo, el valor irrelevante de la celda 6 puede utilizarse para formar un cubo 2 {6, 7, 14, 15} produciendo un implicante primo BC con únicamente dos literales. Sólo una de las irrelevancias se utiliza para formar cubos de orden superior en cada mapa. Nada se gana al utilizar las 8 irrelevancias en la figura 16b o las 6 irrelevancias en la figura 16c. El número de términos y el número de literales en ambas formas son exactamente los mismos. Las expresiones mínimas que se obtienen a partir de los mapas son como sigue:

$$f = wx + xy + w'x'z' \quad y \quad f = (w' + x)(x + z')(w + x' + y)$$

(No lea únicamente esta expresión; verifíquela de manera independiente.) ■

²¹ No hay un símbolo estándar que se utilice universalmente. Otros libros recurren a un guión (–) o d .

Ejercicio 18. Diseñe un circuito combinatorio con cuatro líneas de entrada de manera que la salida f se convierta en 1 siempre que la combinación de entrada $x_3x_2x_1x_0$ represente un número BCD que es igual a una potencia de 2.

- Construya un mapa lógico que cumpla las condiciones del diseño.
- Utilizando el mapa, determine una expresión mínima s de p que represente esta función.
- Dibuje un circuito lógico que implemente esta expresión mínima s de p .
- Utilizando el mapa, obtenga una expresión mínima p de s que represente esta función.
- Dibuje un circuito lógico que implemente esta expresión mínima p de s .
- Advierta las complejidades de cada circuito.

Respuesta²²

7 COMPARADORES

Hasta este momento, en la fase de "implementación" del diseño de circuito combinatorio, los enunciados de los problemas han tenido una connotación "teórica". Esta sección se enfocará en una importante tarea de cómputo: la comparación de las magnitudes de dos números binarios que tienen la misma longitud. Trabajar lo anterior para un caso general resulta algebraicamente muy complejo. En el caso trivial de dos números de 1 bit, una compuerta OR exclusiva tendrá una salida de 0 cuando los 2 bits sean iguales, y de 1 cuando sean diferentes. (Una salida XNOR sería lo opuesto, desde luego.) En cualquier caso, si estos son diferentes, no sabremos cuál es mayor. Conformaremos el caso general considerando primero el de dos números de 2 bits.

Comparadores de 2 bits

Sea $X = x_1x_0$ y $Y = y_1y_0$ los dos números de 2 bits. El objetivo es comparar estos números y determinar sus magnitudes relativas. Vamos a definir las salidas:²³

$$G = (X > Y)$$

$$E = (X = Y)$$

$$L = (X < Y)$$

Se definirá una notación similar para los bits individuales (por ejemplo, E_i significa $x_i = y_i$). Una función lógica de dos variables que se vuelve 1 cuando las variables son iguales se definió en el capítulo 2 como la relación de equivalencia XNOR, el complemento de la OR exclusiva. Así,

$$E_i = (x_i \Leftrightarrow y_i) = (x_i \oplus y_i)' = (x_i y_i' + x_i' y_i)' = x_i y_i + x_i' y_i', \quad i = 1, 0 \quad (1)$$

²² a. La lista de minterminos, incluyendo valores irrelevantes es $f = \Sigma(1, 2, 4, 8) + \Sigma d(10, 11, 12, 13, 14, 15)$. Su mapa debe incluir 1s en las celdas 1, 2, 4, 8 e irrelevancias en las celdas de la 10 a la 15.

b. Ninguno de los minterminos se combina con otros para formar cubos. Recurra a valores irrelevantes para formar cubos 1 con los minterminos 2, 4 y 8. No hay cubos de orden superior. El mintermino 1 (20) no forma ningún cubo k con otros minterminos o con valores irrelevantes. Así, la expresión mínima s de p es $f = x_2'x_1x_0' + x_2x_1'x_0' + x_3x_1'x_0' + x_3'x_2'x_1'x_0$.

c. En el primer nivel están tres compuertas AND de 3 entradas y una de 4 entradas, con una compuerta OR de cuatro entradas en el segundo nivel.

d. Hay seis maxiterminos, correspondientes a los números decimales que no están en la lista de minterminos. El maxitermino 0000 no se combina con otros maxiterminos o valores irrelevantes. El maxitermino 1101 se combina con tres irrelevancias para formar un cubo 2. El maxitermino 0111 se combina por separado con cada uno de los otros tres maxiterminos y dos valores irrelevantes para formar otros tres cubos 2.

La expresión p de s es $f = (x_3' + x_2' + x_1' + x_0')(x_3 + x_0)(x_2 + x_0)(x_1 + x_0)(x_2 + x_1)$.

e. En el primer nivel se encuentran cuatro compuertas OR de 2 entradas y una de 4 entradas, con una compuerta AND de 5 entradas en el segundo nivel.

²³ G significa "mayor que", etcétera.

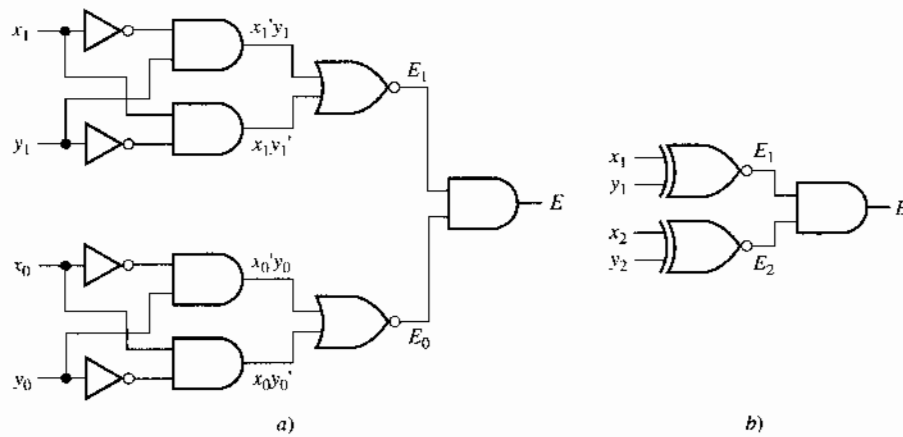


Figura 17. Implementación de un circuito que muestra la igualdad de dos números de 2 bits.

$E_i = 1$ sólo cuando ambos bits correspondientes son iguales: $x_i = y_i$. Entonces E es la AND de todas las E_i ; en el caso presente, $E = E_1 \cdot E_0$, y $E = 1$ cuando $x_1 = y_1$ y $x_0 = y_0$. En una implementación de circuito, una vez que se han obtenido las E_i , una compuerta AND con las E_i como entradas producirá E .

Ejercicio 19

- Construya un diagrama lógico cuyas entradas sean x_1, x_0 e y_1, y_0 y cuya salida corresponda a E . Recurra a la expresión a la derecha en (1) y también a $E = E_1 \cdot E_0$, y suponga que todos los bits se disponen en paralelo. Aunque podría resultar tentador ver el diagrama incluido en la figura 17, consulte la figura únicamente para confirmar lo que usted mismo ha determinado.
- En la figura 17b, suponga que las compuertas XNOR (equivalencias) se sustituyen por compuertas XOR. ¿Qué otro cambio compensatorio tendría que efectuarse?

Respuesta²⁴

De manera similar, la condición $G = 1$ se determina comparando bits correspondientes en las dos palabras, empezando desde el más significativo. Si los bits más significativos son los mismos, se comparan los siguientes bits (y así sucesivamente, de modo conceptual, para palabras de longitud mayor hasta que los 2 bits correspondientes sean distintos "conceptualmente", debido a que éste no es el procedimiento que se usará para números que tengan más de 2 bits).

- Si el bit x es mayor que el bit y , entonces $G = 1$, sin que importen los bits restantes. En este caso, si $x_1 > y_1$ (es decir, $x_1 = 1$ e $y_1 = 0$), entonces $G = 1$, independientemente de x_0 e y_0 .
- Si $x_1 < y_1$, entonces G no puede ser 1; en este caso $L = 1$, sin importar x_0 e y_0 .
- Si $x_1 = y_1$, entonces examinamos el siguiente bit menos significativo; en este caso G puede ser 1 sólo si $x_1 > y_1$. (Es posible establecer un argumento completamente paralelo en relación con L ; efectúelo de manera explícita.)

²⁴ Las entradas a la compuerta AND tendrían que complementarse, por lo que $E = E_1' E_0' = (E_1 + E_0)'$. Sustituya la compuerta AND por una NOR.

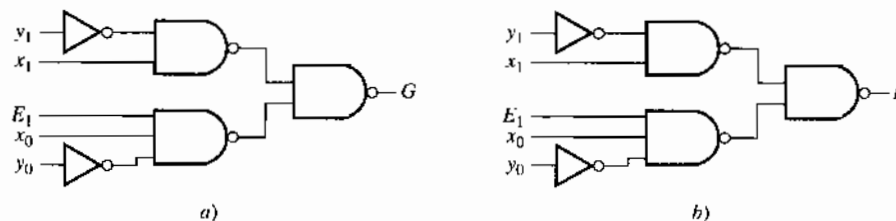


Figura 18. Circuitos parciales para a) G y b) L .

Ejercicio 20. Construya un mapa lógico de cuatro variables para G utilizando $X = x_1x_0$ e $Y = y_1y_0$ como las variables. $G = 1$ en los cuadrados para los cuales $x_1x_0 > y_1y_0$. Escriba una expresión nominal para G utilizando el cubo 2 y los dos cubos 0, factorizando cualquier factor común en los últimos dos.

Respuesta²⁵

Ejercicio 21. Repita el ejercicio 20, esta vez para L .

Respuesta²⁶

Como paso final, advierta que la señal E_1 está disponible en la figura 17 como la salida de una compuerta NOR (o una compuerta XNOR en la figura 17b); las salidas de la compuerta AND $x_1'y_1$ y las otras en las expresiones para G y L también están disponibles en esa figura. Utilizando éstas como entradas, se muestra un diagrama para la implementación de las salidas G y L en la figura 18. (Confirme que los circuitos AND-OR implicados por las expresiones en los pies de página 25 y 26 pueden realizarse mediante circuitos todo NAND, como se indica.) Es posible combinar los dos diagramas en un solo circuito con las cuatro entradas x e y y las tres salidas E , G y L . Efectúe este paso por cuenta propia.

Generalización

Lo que se ha hecho hasta el momento podría considerarse trivial puesto que únicamente se han comparado números de dos bits. Para longitudes de palabras mayores, resultan posibles diferentes procedimientos dependiendo de si el número de bits es par o impar. Considere primero el caso par de cuatro bits.

Comparadores de 4 bits

Es posible obtener los comparadores de cuatro bits utilizando los resultados de los dos comparadores de 2 bits, uno para los 2 bits de orden superior y uno para los 2 bits de orden inferior. Si los dos pares de bits de orden superior no son iguales (un par es mayor que el otro), entonces no es necesario comparar los bits de orden inferior; la decisión se basará por completo en los dos bits de orden superior. (Intente esto para algunos ejemplos, digamos, $10x_1x_0$ y $11y_1y_0$. Demuéstrese a

²⁵ $G = x_1y_1' + x_0y_0'(x_1y_1 + x_1'y_1') = x_1y_1' + x_0y_0'E_1$, donde $E_1 = x_1y_1 + x_1'y_1'$ es la salida de una compuerta XNOR como se muestra en la figura 17b. De acuerdo con (13) en el capítulo 2 y la explicación subsecuente también es igual a $(x_1 \oplus y_1)' = (x_1 \oplus y_1)' = (x_1y_1' + x_1'y_1)'$. Si $E_1 = 0$, eso quiere decir que los primeros bits de los dos números no son iguales. El primer término muestra entonces que la única manera de que G pueda ser 1 es para $x_1 = 1$ e $y_1 = 0$, independientemente de los segundos bits. Si $E_1 = 1$, los primeros bits de los dos números deben ser iguales; por consiguiente, el primer término en G es 0 y G se reduce a x_0y_0' . La única manera de que G pueda ser 1 implica que $x_0 = 1$ e $y_0 = 0$, confirmando de esa manera que $X > Y$.

²⁶ $L = x_1'y_1 + x_1'x_0'y_1'y_0 + x_1x_0'y_1y_0 = x_1'y_1 + x_0'y_0E_1$. Efectúe un análisis como en el pie de la página 23 para confirmar que esta expresión produce el resultado correcto.

usted mismo que las x y las y en la posición cuarta no importan; el segundo número será siempre mayor que el primero, ya que los primeros 2 bits son mayores.) De modo que un comparador de 2 bits es necesario para confrontar los 2 bits de orden superior. Si $G = 1$ o $L = 1$ en este comparador, eso resuelve la pregunta; el par de bits de orden inferior resulta irrelevante. Sólo si los pares de bits de orden superior para cada comparador de 2 bits son los mismos, será necesario comparar los 2 bits de orden inferior. Sin embargo, ese caso requiere también un comparador de 2 bits; los resultados de este comparador determinarán cuál número de 4 bits es mayor. Aún queda un importante problema de realización. La operación completa implicaría sólo uno o a lo más dos comparadores de 2 bits. Investigue los detalles en los problemas.

Comparadores de números pares de bits

Es posible obtener comparadores de cualquier número par de bits de una manera similar. Los pares de bits de orden superior se consideran primero; si $G = 1$ o $L = 1$, eso determina el asunto sin ninguna necesidad de verificar los bits de nivel inferior. Los siguientes pares de nivel más alto se tratan de la misma forma. Sólo en el caso de $E = 1$ en cualquier par de bits será necesario verificar los siguientes pares inferiores. La única vez que resulta necesario verificar los pares de nivel inferior es cuando todos los bits precedentes en los dos números son los mismos. (Revise los detalles.)

Comparadores de números impares de bits

Es factible obtener comparadores de dos números $A = Xa_0$ y $B = Yb_0$ que tienen un número impar de bits poniendo el lsb de los dos números uno al lado del otro y construyendo primero un comparador del número par precedente de bits, X e Y . Recuerde que este comparador tendrá tres salidas. Si la salida G representa $X > Y$, entonces, sin que importen los bits menos significativos, $A > B$. Por otro lado, si L representa $X < Y$, entonces, sin que importen los bits menos significativos, $A < B$. Sólo si $X = Y$ (significando $E = 1$) los bits menos significativos entran en escena. Así, será necesaria circuitería adicional sólo si $X = Y$. En ese caso resultará necesario un procedimiento similar al que se efectuó para dos números de un solo bit, donde los dos números que se van a comparar son a_0 y b_0 . Los detalles se dejan al lector en uno de los problemas.

8 DETERMINACIÓN DEL IMPLICANTE PRIMO: MÉTODO TABULAR

La determinación de expresiones mínimas de suma de productos o de producto de sumas para una función de conmutación determinada puede volverse difícil en el caso de funciones de más de 5 o 6 variables utilizando los métodos que se describieron antes en este capítulo. Lo que se requiere es un procedimiento sistemático que se especifique claramente de manera que sea posible codificarlo en un algoritmo; en ese caso, puede programarse y efectuarse por medio de una máquina. Un procedimiento de minimización de estas características, usualmente denominado el *algoritmo de Quine-McCluskey*, se describirá a continuación.²⁷ Si bien no se explicará aquí un

²⁷ La eliminación de unas cuantas compuertas de un diseño quizá produzca sólo beneficios marginales en la reducción de costo, ya que los diseños de circuitos pequeños se realizan por lo común con circuitos SSI en los cuales se empaquetan en conjunto compuertas primitivas múltiples del mismo tipo y el mismo factor de carga de entrada. Sin embargo, se presenta un beneficio intelectual al comprender los conceptos implicados; ello conduce a un entendimiento más profundo de la estructura de los circuitos lógicos. Además, los circuitos "grandes" casi siempre se implementan mediante "dispositivos lógicos programables" (PLD) preempaquetados, que se explicarán en el capítulo siguiente. La estructura interna de estos dispositivos corresponde a una AND-OR de dos niveles, con un número fijo de AND. Si una función no minimizada tiene demasiadas AND, incluso es posible para implementarla usar un PLD disponible. Por consiguiente, la minimización constituye una herramienta esencial incluso en el caso de funciones con gran número de variables. Además, los conceptos formulados en el contexto de la minimización de una expresión de conmutación pueden utilizarse también en otros contextos —por ejemplo, minimizar un experimento para el diagnóstico de fallas en circuitos lógicos, un área avanzada que no se aborda en este libro.

programa de computadora para efectuar el algoritmo, tales programas están disponibles.²⁸ En el proceso de minimización se incluyen dos pasos:

- Determinar todos los implicantes primos.
- Determinar un conjunto mínimo de éstos que cubra a la función.

Cada proceso es algorítmico.

La idea básica en el método que se va a describir es que dos minitérminos lógicamente adyacentes que son productos canónicos de literales (cubos 0 sobre un mapa) pueden combinarse en un producto en el cual falta una de las literales (un cubo 1 sobre el mapa).

Además, dos cubos 1 que son adyacentes —esto es, dos productos de literales que son los mismos excepto porque la literal *i*ésima aparece como x_i en un producto y como x_i' en el otro— se pueden combinar en un cubo 2. Este proceso continúa hasta que no hay más cubos k adyacentes. El conjunto de *todos* los implicantes primos se representa por medio de un conjunto de todos los cubos k que no están cubiertos por completo por un cubo k de orden superior.

Suponga que un mapa de cuatro variables incluye los siguientes minitérminos, así como otros: {4, 5, 12, 13}. Tenderíamos a encerrar los cuatro minitérminos en un cubo 2 sin advertir necesariamente que los minitérminos forman *cuatro* cubos 1 distintos, todos cubiertos por el cubo 2 para estar seguros. Si buscamos un método tabular libre de problemas donde se forman cubos de orden superior a partir de aquéllos del siguiente orden inferior, no podemos dejar los cubos k al capricho de una identificación aleatoria. Debemos considerar de manera sistemática *todas* las adyacencias posibles.

Representaciones de cubos k adyacentes

Como usted sabe, las representaciones binarias de dos celdas adyacentes difieren exactamente en una posición de bit. Considere la función: $f = \Sigma (2, 3, 4, 5, 7, 8, 10, 11, 12, 13)$ con el conjunto de minitérminos {4, 5, 12, 13}, es posible formar los siguientes cubos k :

cubo 1 {4, 5}	cubo 1 {12, 13}	cubo 2 {4, 5, 12, 13}
4 0100	12 1100	{4, 5} (010-)
(010-)	(110-)	(-10-)
5 0101	13 1101	{12, 13}(110-)

Cada posición de bit en un número binario corresponde a una potencia de 2. Cuando dos cubos k adyacentes forman un cubo $(k + 1)$, la posición en la cual difieren los valores binarios se sustituye por medio de -, como ya se mostró.

Surge la siguiente pregunta: si las representaciones binarias de dos minitérminos adyacentes difieren exactamente en una posición, ¿cómo difiere su representación decimal? Puesto que cada posición corresponde a una potencia de 2, y como el bit en esta posición es 1 en un minitérmino y 0 en el otro, la diferencia en las dos representaciones decimales es exactamente esta potencia de 2. Esto es:

Si dos minitérminos son adyacentes, sus números minitérminos decimales difieren por una potencia de dos.

²⁸ En cuanto a una descripción del programa en lenguaje Pascal para efectuar el algoritmo de Quine-McClusky, consulte las páginas 236-243 de Wakerly, *Digital Design*. (Véase la bibliografía.) El proceso que se describe en esta sección es algorítmico y repetitivo; podría considerarse tedioso. Puesto que existe un programa de computadora para el algoritmo que se analiza, quizá desee omitir la sección. Sin embargo, ésta resulta intelectualmente satisfactoria y al menos valdría la pena profundizar algo en ella.

Índice (Núm. de 1s)	Cubos 0 (minitérminos)	Cubos 1	Cubos 2	Implicantes primos
1	2✓ 4✓ 8✓	2, 3 (1)✓ 2, 10 (8)✓ 4, 5 (1)✓ 4, 12 (8)✓ 8, 10 (2) P_3 8, 12 (4) P_4	2, 3, 10, 11, (1, 8) P_1 4, 5, 12, 13, (1, 8) P_2	$P_1 = \{2, 3, 10, 11\}$ $P_2 = \{4, 5, 12, 13\}$ $P_3 = \{8, 10\}$ $P_4 = \{8, 12\}$ $P_5 = \{3, 7\}$ $P_6 = \{5, 7\}$
2	3✓ 5✓ 10✓ 12✓	3, 7 (4) P_5 3, 11 (8)✓ 5, 7 (2) P_6 5, 13 (8)✓		
3	7✓ 11✓ 13✓	10, 11 (1)✓ 12, 13 (1)✓		

a)

b)

Figura 19. Determinación tabular de los implicantes primos de $f = \Sigma(2, 3, 4, 5, 7, 8, 10, 11, 12, 13)$.

¿Lo inverso también es cierto? Esto es, si los números de minitérminos difieren por una potencia de 2, ¿los minitérminos son adyacentes? ¡la respuesta es no! Como un ejemplo, los minitérminos 10 (1010) y 6 (0110) difieren por $4 = 2^2$, una potencia de 2; pero no son adyacentes. Sus valores binarios difieren en dos posiciones, no en una sola. Este hecho tiene implicaciones para el procedimiento que estamos en proceso de describir; significa que la prueba para la adyacencia nunca debe efectuarse en forma decimal, sino siempre en representación binaria.

Clasificación por índice

Si los valores binarios de dos minitérminos difieren en una posición, eso significa que el número de bits con valor 1 difiere en 1. Dejemos que el *índice* de un minitérmino se defina como el número de 1s en su representación binaria. Eso quiere decir que si los minitérminos se clasifican por índice, entonces sólo aquéllos cuyos índices difieren en 1 pueden posiblemente ser adyacentes. ¿Pero qué pasa con lo inverso? Si los índices de dos minitérminos difieren en 1, ¿es necesariamente cierto que los valores binarios difieren en una *posición*? La respuesta es otra vez negativa. Los índices de los minitérminos 8 y 3, por ejemplo, son 1 y 2, respectivamente. Incluso sus valores binarios difieren en tres posiciones. Así, la condición de que los números decimales de dos minitérminos difieren por una potencia de 2 y la condición de que el número de 1s en sus representaciones binarias difieren en 1 constituyen condiciones *necesarias*, aunque *no suficientes*.

Vamos a regresar al procedimiento tabular para una función dada. El primer paso consiste en listar los números de los minitérminos, agrupados por índice, en la primera columna de una tabla. El ejemplo que se presentó en la subsección precedente se usa en la tabla de la figura 19.

El siguiente paso es probar aquellos minitérminos que *podrían* ser adyacentes de acuerdo con sus índices para ver si en realidad lo son.

Cada número de minitérmino en el grupo con índice 1 se verifica contra cada uno en el grupo con índice 2. Los cubos 1 formados por dos cubos 0 adyacentes se identifican listando los pares de números de minitérmino seguidos (en paréntesis) por el valor de la potencia de 2 en el cual difieren sus valores binarios, como se muestra en la tercera columna. El cubo 1 formado por los minitérminos 4 y 12, con diferencia de 8, por ejemplo, se indica como 4, 12 (8). Puesto que

aquellos cubos 0 son cubiertos por un cubo k de orden superior y, en consecuencia, no pueden ser implicantes primos, se eliminan.

Advierta que aun cuando el minitérmino 4 (índice 1) difiere del minitérmino 3 (índice 2) por una potencia de 2, m_4 y m_3 no son adyacentes: difieren en tres posiciones binarias. Además, 4 en un grupo de índices inferior es un número mayor que 3 en el siguiente grupo de índice superior. La generalización de estas observaciones es la siguiente: no hay necesidad de verificar la adyacencia de un minitérmino que tiene un índice inferior con otro que tiene un índice superior si el último es un número más pequeño; éstos no pueden ser adyacentes.

Cuando se forman todos los cubos 1 que incluyen minitérminos con índices 1 y 2, entonces los minitérminos de índice 2 se verifican con los minitérminos de índice 3 en cuanto a la adyacencia. Sin embargo, los cubos 1 formados por dichas adyacencias se listan por separado a partir del conjunto previo. El proceso continúa hasta que se agotan todos los índices. En el ejemplo presente hay dos minitérminos con índices superiores, por lo que este paso termina.

Lo que se hace después es evidente: formamos cubos 2 de pares adyacentes de cubos 1. Los números entre paréntesis que siguen a los números de minitérminos en los cubos 1 son las potencias de 2 correspondientes a la posición binaria en la cual difieren los dos minitérminos que forman el cubo 1. El cubo 1 con denominación 4, 12 (8), por ejemplo, tiene la representación binaria -100. El valor binario de 12 tiene un 1 en la posición 8 ($=2^3$), en tanto que el valor binario de 4 tiene un 0 ahí. Si este cubo 1 va a ser adyacente a otro cubo 1, ese otro debe tener también un 8 entre paréntesis, lo que significa un - en la posición 2^3 de su valor binario.

Al mismo tiempo, en el resto de sus bits, los dos cubos 1 deben diferir en exactamente una posición. Estas dos condiciones se confirman en los dos pasos comparando cada cubo 1 en la primera categoría con aquéllos en la siguiente. Primero, los números entre paréntesis deben coincidir; luego el primero de los dos minitérminos en cada cubo 1 debe diferir por una potencia de 2 si los dos cubos 1 van a ser adyacentes. (Si todos los minitérminos se listan en orden ascendente con cada cubo k , entonces es suficiente verificar el primer minitérmino; los otros diferirán automáticamente por la misma potencia de 2.) Empezando con el cubo 1 con denominación 2, 3 (1), por ejemplo, los únicos cubos 1 en el siguiente grupo por verificar son los que tienen (1) entre paréntesis: 10, 11 (1) y 12, 13 (1).

Para verificar si los primeros minitérminos de los pares de cubos 1 difieren por una potencia de 2, advertimos que 2 y 10 lo hacen ($10 - 2 = 8 = 2^3$), aunque 2 y 12 no. De modo que sólo (2, 3, 10, 11) forman un cubo 2. Para identificar las posiciones en las cuales difieren sus bits, escribimos tanto 1 como 8 entre paréntesis, como se muestra en la última columna en la tabla.

Luego, el siguiente cubo 1 -2, 10, (8) en el primer grupo, se verifica con los del siguiente grupo. Los únicos candidatos con los cuales puede formar un cubo 2 son aquéllos con 8 entre paréntesis: 3, 11 (8) y 5, 13 (8). Para 3, 11 (8), los primeros minitérminos (2 y 3) son adyacentes, pero no ocurre lo mismo para 5, 13, (8). El cubo 2 formado de ese modo es 2, 3, 10, 11 (1, 8); es el mismo que el ya encontrado y por ello no necesita listarse de nuevo. En cada paso, los cubos 1 que forman un cubo 2 se verifican como si hubieran sido cubiertos y, en consecuencia, como si no fueran implicantes primos.

El proceso de formar cubos 2 se continúa hasta que todos los cubos 1 se han agotado. En el ejemplo que nos ocupa, sólo existen dos cubos 2. Ocho de los cubos 1 están completamente cubiertos por ellos, dejando cuatro cubos 1 sin cubrir. El total de estos cubos 1 son también implicantes primos, integrando un total de seis, como se indica en la tabla.

La descripción que acaba de presentarse ha sido prolongada, pero el proceso real es más o menos simple. Se repiten unos cuantos pasos una y otra vez; como ya se señaló, el proceso puede implementarse mediante un programa de computadora.

El proceso para encontrar todos los implicantes primos mediante el método tabular se resume en la tabla 1; genera *todos* los implicantes primos de una función de conmutación especificada. A partir de este conjunto, aún necesitamos elegir un conjunto mínimo, con el menor número de literales, que cubra todos los minitérminos. Éste es el tema de la sección subsecuente.

Tabla 1 Resumen de la determinación de implicantes primos

1. Dada una función de conmutación, obtenga una lista de minitérminos; agrupe los números de minitérminos decimales en una columna en orden de índices ascendentes (número de 1s).
2. Verifique la adyacencia de cada minitérmino que tenga un valor de índice inferior con cada uno que tenga el valor de índice superior siguiente, empezando con el índice más bajo; éstos son adyacentes si su valor decimal difiere por una potencia de 2 y su valor binario difiere sólo por una posición de bit. Liste los cubos 1 formados por minitérminos adyacentes, dando ambos números de minitérminos en orden ascendente, y la diferencia de potencias de 2 entre ellos entre paréntesis. Repita hasta que todos los valores de índices se agoten. Elimine cada minitérmino cubierto por un cubo 1 puesto que éste no es un implicante primo. Separe los conjuntos de cubos 1 formados por cada par de índices.
3. Empezando con el par de índices más bajo, repita para los grupos de cubos 1. Confirme la adyacencia de todos los cubos 1 en un grupo formado por un par de índices con aquéllos en el grupo formado por el siguiente par de índices; éstos serán adyacentes sólo si tienen el mismo número entre paréntesis y si los primeros números de minitérminos difieren por una potencia de dos. Liste los cubos 2 formados por cubos 1 adyacentes, indicando todos los números de minitérminos en orden ascendente y, además de los números ya entre paréntesis, la diferencia de potencias de 2 en los primeros minitérminos listados de los dos cubos 1. También en este caso, se eliminan los cubos 1 cubiertos por los cubos 2 para indicar que no pueden ser implicantes primos. Separe los conjuntos de cubos 2 formados por cada grupo de cubos 1.
4. Repita para cada conjunto de cubos k hasta que ya no puedan formarse cubos de orden superior. Todos los cubos k no eliminados representan implicantes primos. Numérelolos, empezando con los de orden más alto.

Funciones incompletamente especificadas

El procedimiento anterior, el cual se aplica a funciones completamente especificadas, se extiende sin dificultad a funciones incompletamente especificadas. Al efectuar el algoritmo para ese caso, todos los valores irrelevantes se toman como 1s. Esto conducirá a un número relativamente grande de implicantes primos, incluyendo algunos que cubren sólo valores irrelevantes. Si bien esto requiere cierto esfuerzo, el paso subsecuente de selección de un conjunto mínimo entre los implicantes primos garantizará que una expresión mínima no incluya a todos estos implicantes primos irrelevantes, como se mostrará en la sección siguiente.

EJEMPLO 7

Dada la siguiente función, el objetivo es generar todos los implicantes primos:

$$f(A, B, C, D, E) = \Sigma(0, 4, 5, 7, 9, 12, 13, 14, 15, 23, 31) + \Sigma d(3, 6, 10, 16, 20)$$

La figura 20 muestra la tabla que se construye listando todos los valores irrelevantes como si fueran 1s de acuerdo con sus índices.

Puesto que 0 es un minitérmino en este ejemplo, también aparece el índice 0. Se lleva a cabo el proceso indicado en la tabla 1. Todos aquellos cubos k sin verificar son implicantes primos. Confirme todos los detalles de esta tabla. ■

Selección de una expresión mínima

Luego de que se han determinado los implicantes primos, el siguiente paso es seleccionar el número más pequeño de ellos que en conjunto cubren todos los minitérminos y tienen el menor número de literales.

Índice	Cubos 0	Cubos 1	Cubos 2	Cubos 3
0	0✓	0, 4 (4)✓ 0, 16 (16)✓	0, 4, 16, 20, (4, 16) P_2	4, 5, 6, 7, 12, 13, 14, 15 (1, 2, 8) P_1
1	4✓ 16✓	4, 5 (1)✓ 4, 6 (8)✓	4, 5, 6, 7, (1, 2)✓ 4, 6, 12, 14, (2, 8)✓ 4, 5, 12, 13, (1, 8)✓	
2	3✓ 5✓ 6✓ 9✓ 10✓ 12✓ 20✓	4, 12 (8)✓ 4, 20 (16)✓ 16, 20 (4)✓	5, 7, 13, 15, (2, 8)✓ 6, 7, 14, 15, (1, 8)✓ 12, 13, 14, 15, (1, 2)✓	
3	7✓ 13✓ 14✓	3, 7 (4) P_4 5, 7 (2)✓ 5, 13 (8)✓ 6, 7 (1)✓ 6, 14 (8)✓ 9, 13 (4) P_5 10, 14 (4) P_6 12, 14 (2)✓	7, 15, 23, 31 (8, 16) P_3	
	15✓ 23✓	7, 15 (8)✓ 7, 23 (16)✓		
	31✓	13, 15 (2)✓ 14, 15 (1)✓		
		15, 31 (16)✓ 23, 31 (8)✓		

Figura 20. Procedimiento tabular para la determinación de implicantes primos de la función $f = \Sigma(0, 4, 5, 7, 9, 12, 13, 14, 15, 23, 31) + \Sigma d(3, 6, 10, 16, 20)$.

Este proceso se simplifica construyendo una *tabla o diagrama de implicantes primos*, en la cual los implicantes primos constituyen los renglones y los minitérminos, las columnas. Si el implicante primo P_j cubre al minitérmino m_i , se anota una marca de verificación en la intersección del renglón correspondiente a P_j y la columna correspondiente a m_i .

Funciones completamente especificadas

Debido a que hay diferencias, aunque pequeñas, tratamos primero funciones completamente especificadas, empezando con un ejemplo.

EJEMPLO 8

Los implicantes primos para $f = \Sigma(2, 3, 4, 5, 7, 8, 10, 11, 12, 13)$ se establecieron en la figura 19. El diagrama de implicantes primos resultante se presenta en la figura 21. Los implicantes primos de orden superior se listan primero y el conjunto de cubos 2 se separa de los cubos 1. Advierta que el número de minitérminos cubierto por un cubo k es 2^k . En esta forma en que muchas marcas de verificación deben estar en un renglón en el caso de una función específica. Confirme todos los detalles de esta tabla.

Puesto que cada implicante primo esencial cubre un minitérmino que no cubre ningún otro implicante primo, un conjunto mínimo *debe* incluir los implicantes primos esenciales. Para iden-

	2	3	4	5	7	8	10	11	12	13
P_1	✓	✓					✓	✓		
P_2			✓	✓					✓	✓
P_3						✓	✓			
P_4						✓			✓	
P_5		✓			✓					
P_6				✓	✓					

a)

$$\begin{aligned}
 P_1 &= B'C \\
 P_2 &= BC' \\
 P_3 &= AB'D \\
 P_4 &= AC'D' \\
 P_5 &= A'CD \\
 P_6 &= A'BD
 \end{aligned}$$

	2	3	4	5	7	8	10	11	12	13
* P_1	⊙	✓					✓	⊙		
P_2			⊙	✓					✓	⊙
P_3						✓	✓			
P_4						✓			✓	
P_5		✓			✓					
P_6				✓	✓					

b)

Figura 21. Tabla de implicantes primos para $f = \Sigma(2, 3, 4, 5, 7, 8, 10, 11, 12, 13)$.

tificar a éstos, cada columna se explora verticalmente. Si hay una columna con una sola marca de verificación, esa columna corresponde a un minitérmino distinguible, y el renglón en el cual aparece es un impicante primo esencial. Como se ilustra en la figura 21b, cuando un minitérmino se cubre mediante un impicante primo que se selecciona, se pone un círculo alrededor de la marca de verificación.

En la figura 21 existen cuatro minitérminos distinguibles pero sólo dos implicantes primos esenciales, indicados por asteriscos. Los minitérminos cubiertos por los implicantes primos esenciales no necesitan ser cubiertos por cualquier otro impicante primo. Para identificar los minitérminos ya cubiertos, se coloca una marca de verificación sobre aquellos números de minitérmino en el encabezado de las columnas. Los únicos minitérminos que no cubren P_1 y P_2 son 7 y 8. La exploración de esas columnas indica que el minitérmino 7 es cubierto por P_5 y P_6 , y el minitérmino 8, ya sea por P_3 o P_4 .

Puesto que el número de literales en cada uno de éstos es el mismo, la selección de cualquier combinación de estos implicantes primos conducirá a una expresión mínima:

$$\begin{aligned}
 f &= P_1 + P_2 + P_3 + P_5 = P_1 + P_2 + P_3 + P_6 \\
 &= P_1 + P_2 + P_4 + P_5 = P_1 + P_2 + P_4 + P_6
 \end{aligned}$$

La forma de producto de literales para un impicante primo correspondiente a un cubo k se obtiene de la siguiente manera. Se escribe primero el valor binario de cualquiera de los números de minitérmino cubiertos por el cubo k ; luego se reemplaza con $-$ el bit en la posición donde uno de los otros minitérminos en el cubo k tiene el valor complementario. De ese modo, en el cubo 1 $P_4 = (8, 12)$ el valor binario para 8 es 1000. El valor binario correspondiente a 12 difiere de éste sólo en la posición 2^2 , dando lugar a 1-00. Suponiendo que las variables son A, B, C, D , la segunda variable, B , falta en el impicante primo, y el producto es $AC'D'$. Verifique las expresiones para los otros implicantes primos dados en la figura. ■

La tarea de minimizar los circuitos de salida múltiple se deja por lo general a las herramientas de diseño asistido por computadora. Ese tema está más allá del objetivo de este libro, y no lo volveremos a considerar. A pesar de eso, la mayor parte de los circuitos que se presentan en los capítulos siguientes son de salida múltiple por naturaleza. Entre éstos se encuentran los dispositivos lógicos programables que se describen en el capítulo 4.

RESUMEN Y REPASO DEL CAPÍTULO

Este capítulo dispuso los fundamentos para métodos con los que se representan e implementan funciones lógicas. Las realizaciones consideradas aquí son aquellas que utilizan compuertas primitivas, las precursoras de los circuitos MSI y las realizaciones con PLD, que se explicarán en los capítulos siguientes. Se incluyeron los siguientes temas.

- Minitérminos, listas de minitérminos, y realizaciones de suma de productos.
- Maxitérminos, listas de maxitérminos, y forma de producto de sumas.
- Mapas lógicos (Karnaugh) y adyacencia.
- Adyacencia lógica y adyacencia de mapa.
- Minitérminos de agrupamiento sobre un mapa.
- Cubos de orden k , o cubos k .
- Expresiones lógicas irreducibles y mínimas.
- Una función que cubre otra.
- Implicantes de una función de conmutación.
- Implicantes primos.
- Implicantes primos esenciales.
- Expresiones equivalentes.
- Expresiones mínimas de suma de productos.
- Expresiones mínimas de producto de sumas.
- Implementaciones de funciones de conmutación:
 - Realizaciones de dos niveles.
 - Realización AND-OR.
 - Realización NAND.
 - Realización OR-AND.
 - Realización NOR.
 - Realizaciones multinivel.
- Análisis de circuitos lógicos.
- Funciones incompletamente especificadas y valores irrelevantes.
- Comparadores de magnitud.
- Determinación de implicantes primos: Algoritmo de Quine-McCluskey.
 - Clasificados por índice: funciones completamente especificadas.
 - Funciones incompletamente especificadas.
 - Obtención de una expresión mínima.
- Diagramas de temporización.
- Riesgos.
- Circuitos sin riesgo.

PROBLEMAS

Recuerde guardar las soluciones de las primeras partes de los problemas de modo que pueda referirse a ellas cuando esté listo para efectuar los problemas de las partes finales.

1. Para cada tabla de verdad dada en la figura P1:
 - a. Construya la lista de minitérminos.
 - b. Construya el mapa lógico (mapa K).
 - c. Escriba la expresión canónica de suma de productos.
 - d. Especifique todos los implicantes primos e indique todos los que sean esenciales.
 - e. Determine al menos una expresión mínima s de p a partir del mapa.
 - f. Escriba la expresión canónica del producto de sumas.
 - g. Encuentre al menos una expresión mínima p de s utilizando el mapa.

x	y	z	f_1	f_2	f_3	f_4
0	0	0	1	1	0	1
0	0	1	1	0	1	1
0	1	0	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	1	1	0	0	0

Figura P1

2. Construya un mapa lógico para cada una de las siguientes funciones.

- a. $f_1 = y + xy' + y'z$
- b. $f_2 = xy + x'z' + yz'$
- c. $f_3 = ABD' + A'BD + A'C'D + AC + BCD + B'C'D$
- d. $f_4 = A'B + A'CD' + AC'D + B'CD' + BC'D$

En cada caso, encuentre una expresión mínima de suma de productos.

3. Dos variaciones de un mapa lógico de tres variables se proporcionaron en la figura 3. Ahora es su turno.

- a. Determine un orden de las variables $wxyz$ en un mapa de cuatro variables, y el orden de sus valores lógicos, tales que cada celda sea adyacente a las otras cuatro.
- b. Dada $f = (wx' + y'z)(x + w'y')$, construya un mapa lógico utilizando el patrón de la parte a.
- c. A partir del mapa, determine las expresiones mínimas s de p y p de s que represente a esta función.

4. Se proporciona la siguiente función de cuatro variables:

$$f = \Sigma(0, 3, 4, 6, 7, 9, 12, 13, 14, 15)$$

- a. Con el uso de un mapa, encuentre el conjunto de todos los implicantes primos y especifique aquellos que son esenciales.
- b. Especifique las celdas 1 en cada mapa y encuentre las expresiones mínimas s de p .
- c. Repita las partes a y b para el complemento f' .
- d. Convierta la expresión para f' en una expresión para f :
 - i. Utilizando el teorema de De Morgan una vez.
 - ii. Utilizando el teorema de De Morgan dos veces.
- e. Repita las partes de la a a la d para cada una de las siguientes funciones:

$$f_1 = \Sigma(1, 5, 6, 9, 10, 11, 12, 13)$$

$$f_2 = \Sigma(1, 2, 4, 6, 7, 14, 15)$$

$$f_3 = \Sigma(0, 3, 5, 7, 11, 13, 14, 15)$$

$$f_4 = \Sigma(1, 2, 4, 6, 8, 9, 10, 11)$$

- f. Obtenga la realización de suma de productos de cada una de las funciones anteriores.
- g. Obtenga la realización del producto de sumas de cada función.

5. Se indica la siguiente función de conmutación:

$$f(A, B, C, D) = \Sigma(0, 1, 2, 4, 6, 10, 11, 12, 13, 14)$$

- a. Con el uso de un mapa lógico, encuentre todos los implicantes primos y especifique todos los que sean esenciales.
- b. Encuentre todas las expresiones mínimas distintas para f que sea posible.
- c. Determine expresiones mínimas para el complemento f' , directamente del mapa.
- d. Repita las partes anteriores para cada una de las funciones en el problema 4.

6. Se indica la siguiente función de cuatro variables:

$$f = \Sigma(1, 3, 5, 6, 7, 9, 11, 12, 13)$$

- a. Con el uso de un mapa lógico, determine todas las expresiones mínimas s de p posibles.
- b. Repita para el caso de expresiones p de s . ¿Cuál realización tiene menor número de términos y literales?
- c. A partir del mapa, encuentre una expresión mínima s de p para f' .

7. Mediante el uso de un mapa lógico, determine expresiones mínimas s de p para las siguientes funciones. En cada caso, encuentre las celdas 1 distinguibles en el mapa.

$$a. f_1 = \Sigma(0, 2, 3, 4, 6, 8, 10, 11, 12, 14)$$

$$b. f_2 = \Sigma(0, 4, 6, 8, 12, 14)$$

$$c. f_3 = \Sigma(1, 3, 7, 9, 12, 13, 14, 15)$$

$$d. f_4 = \Sigma(0, 2, 6, 7, 8, 9, 13, 15)$$

$$e. f_5 = \Sigma(4, 5, 6, 9, 11, 13, 14, 15, 20, 21, 22, 25, 27, 29, 31)$$

Determine también una expresión mínima s de p para el complemento de cada función.

8.
 - a. Utilizando un mapa lógico, encuentre todas las expresiones mínimas p de s para las funciones en el problema 7.
 - b. Repita para el complemento de cada función.
9.
 - a. Una función de conmutación de cuatro variables $f(w, x, y, z)$, es igual al producto de otras dos funciones, f_1 y f_2 , de las mismas variables: $f = f_1 f_2$. Se indican las funciones f y f_1 :

$$f = \Sigma(4, 7, 15)$$

$$f_1 = \Sigma(0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 15)$$

Determine el número de funciones completamente especificadas f_2 (sin valores irrelevantes) que satisfarán las condiciones dadas. Elija la más simple de estas funciones.

- b. Se va a diseñar un circuito lógico para realizar la siguiente función f .

$$f(A, B, C, D) = A'B + B'C + BD'$$

Ya sea manipulando esta expresión o a partir de mapas lógicos de f o f' , llegue a una forma cuya realización incluya cualquier número de compuertas AND y OR, pero únicamente un inversor sencillo.

- c. Una función lógica f tiene n implicantes primos, P_1, \dots, P_n . Se encuentra que el producto de par amplio $P_i P_j = 0$ para todos los pares. Demuestre que la función tiene una expresión mínima única de suma de productos.
- d. Una función de cuatro variables va a tener ocho minitérminos y ningún implicante primo esencial. Muestre un mapa lógico posible para dicha función.

10. Utilizando el método tabular, determine expresiones mínimas s de p para f .
- En el problema 4.
 - En el problema 5.
11. a. Repita el problema 6 utilizando el método tabular.
b. Repita el problema 7 utilizando el método tabular.
12. El mapa lógico en la figura P12 no tiene implicantes primos esenciales.
- Determine los mapas de otras dos de tales funciones de cuatro variables que tengan el mismo número de minterminos. Encuentre una expresión mínima s de p en cada caso.
 - Determine el mapa de una función de cuatro variables con nueve minterminos que tenga la misma propiedad.

		wx			
		00	01	11	10
yz	00	1	1	1	
	01			1	
	11	1	1	1	
	10	1			

Figura P12

13. Dada la función incompletamente especificada de cuatro variables

$$f = \Sigma(0, 1, 2, 4, 7, 12) + \Sigma d(8, 10, 15)$$

- Encuentre una expresión mínima s de p .
 - Encuentre una expresión mínima p de s .
 - Emplee la ley distributiva para convertir la respuesta de la parte b en una expresión s de p . Explique por qué no es la misma que la respuesta de la parte a .
14. Se dan las siguientes funciones completamente especificadas. Con el uso de un mapa lógico, encuentre expresiones mínimas s de p y p de s para cada una.
- $f_1 = \Sigma(2, 4, 6, 10, 12) + \Sigma d(0, 8, 9, 13)$
 - $f_2 = \Sigma(4, 5, 7, 12, 14, 15) + \Sigma d(3, 8, 10)$
 - $f_3 = \Sigma(0, 6, 7, 10, 12) + \Sigma d(2, 8, 9, 15)$
 - $f_4 = \Sigma(1, 2, 3, 4, 5, 11, 18, 19, 20, 21, 23, 28, 31) + \Sigma d(0, 12, 15, 27, 30)$
 - $f_5 = \Sigma(5, 7, 12, 13, 15, 23, 24, 27, 28, 31) + \Sigma d(8, 16, 18, 21, 26, 29)$
15. Repita el problema 14 utilizando el método tabular.
16. Las cuatro entradas al circuito lógico que se muestran en la figura P16 representan un dígito decimal codificado en binario (BCD) en el orden $x_3x_2x_1x_0$. Se sabe que nunca ocurren esas combinaciones de entrada que no corresponden a una palabra de código BCD para un dígito decimal. La salida z será 1 si y sólo si la palabra de entrada representa un dígito par.
- Complete un mapa lógico para la condición dada.
 - A partir del mapa, determine una expresión mínima s de p .
 - Dibuje una realización AND-OR de dos niveles a partir de este mapa; luego conviértala en un circuito todo NAND.
 - Suponga que sólo se dispone de NAND de dos entradas. Construya un circuito utilizando únicamente estas compuertas.
 - Determine una expresión mínima p de s .

- f. Construya una realización OR-AND de dos niveles a partir del mapa; después conviértala en un circuito todo NOR.
- g. Dibuje una realización de circuito si sólo se dispone de NOR de dos entradas.

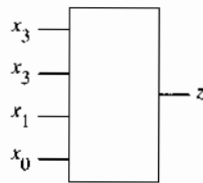


Figura P16

17. Suponga que la palabra de entrada en el problema 16 representa un dígito que es 0 o una potencia de 2. Repita los pasos del a al g para este caso.
18. Suponga que la palabra de entrada en el problema 16 representa un dígito que es primo (0 y 1 no se consideran primos). Repita los pasos del a al g para este caso.
19. Suponga que la palabra de entrada en el problema 16 corresponde al dígito decimal 0, 2, 3, 5 u 8. Repita los pasos del a al g para este caso.
20. En el escenario del problema 16, suponga que las combinaciones prohibidas que representan a los números decimales del 10 al 15 ya no se considerarán así.
- a. Repita el problema 17 para el detector de potencias de 2.
- b. Repita el problema 18 para el detector de números primos.

En cada caso, indique los implicantes primos esenciales y los minitérminos (o maxitérminos) que los distinguen.

21. Suponga que se incrementa a cinco el número de entradas en el problema 16, y que también se incrementa el número de entradas que representa a un número decimal en código binario, no BCD. Repita el problema 18 para el detector de números primos.
22. Un conjunto de circuitos lógicos va a tener un número par de líneas de entrada: cuatro, seis u ocho. La mitad de las entradas se indican mediante x_i , y la otra mitad como y_i . Cada conjunto de entradas x_i representa un número decimal (en código binario $X = x_1x_0, x_2x_1x_0$, o $x_3x_2x_1x_0$), como ocurre con cada conjunto de entradas y_i para formar Y en un orden similar. El propósito de cada circuito es comparar los dos números entrantes X e Y y producir la salida $z = 1$ si y sólo si $X > Y$. Determine las expresiones mínimas s de p y p de s en cada caso.
23. Cierta universidad con exceso de profesores emplea a cinco de ellos para calificar cada examen; cada uno de ellos debe calificar al examen *aprobado* (1) o *reprobado* (0). Para aprobar un examen, un estudiante debe recibir cuatro o cinco aprobaciones. Se reprueba el examen si el estudiante recibe cuatro o cinco reprobaciones. El examen debe volverse a efectuar si el número de aprobaciones es dos o tres. Se va a diseñar un circuito lógico, con cinco entradas x_i y dos salidas y y z . Ambas salidas deben ser iguales a 1 si se aprueba el examen; ambas salidas corresponderán a 0 si el examen se reprueba; las salidas deben ser $yz = 10$ si el examen se va a repetir.
- a. Determine una expresión mínima s de p para z .
- b. A partir de los mapas de y y z , deduzca una relación entre las dos salidas. A partir de esta última, encuentre una expresión p de s para y .
- c. Confirme que esta expresión es mínima.
- d. Utilice la ley distributiva (y cualquier otra) en la parte b para obtener una expresión s de p correspondiente a y .
- e. Confirme mediante el método tabular que ésta es una expresión mínima.
- f. Modifique las expresiones para las dos salidas a fin de formar tantos términos comunes como sea posible, reduciendo de esa manera el número de compuertas tanto como se pueda.
24. La tabla de implicantes primos de una función se muestra en la figura P24. Todo implicante primo de la función se incluye en los renglones de la tabla. Todo minitérmino (para el cual $f = 1$) se lista en los encabezados de columna, aunque dos de ellos no se conocen. (El orden de esos números de minitérmino no es necesariamente como se indica.)

- Encuentre los números de minitérmino m_a y m_b . Si hay más de una posibilidad, indique todas. Muestre un mapa lógico de la función.
- Determine una expresión para cada uno de los implicantes primos no especificados.

Implicantes primos	Minitérminos						m_a	m_b
	1	3	4	6	9			
$P_1 = B'D$	✓	✓			✓			
$P_2 = A'BD'$			✓	✓				
$P_3 =$				✓		✓		
$P_4 =$			✓					✓
$P_5 =$	✓							✓
$P_6 =$					✓			

Figura P24

25. Dada una función representada por una expresión $f(\{x_i\})$, con $i = 1, 2, \dots, n$, defina la *función dual* f_d como la representada por medio de la expresión que se obtiene al intercambiar las operaciones lógicas de suma y multiplicación. (Si las constantes 0 y 1 aparecen explícitamente en la función, éstas también se intercambian en el dual. Sin embargo, si consideramos que la función no tiene redundancias, las constantes no aparecerán de manera explícita.) Demuestre lo siguiente:

$$f_d = f'(\{x_i'\})$$

Esto es, el dual de la función surge de complementar la expresión que la representa y de complementar también cada literal en la expresión. (En este proceso, se debe prestar la debida atención para asegurar que se respeten los paréntesis implicados.)

- Una función de conmutación de tres variables tiene minitérminos m_3 y m_5 . Si se complementan las literales en estos minitérminos, ¿cuáles son los números de minitérminos correspondientes? Repita para el caso de una función de cuatro variables con minitérminos m_6 y m_{13} . Generalice estas observaciones para una función con n variables. Esto es, ¿cuál es el número del minitérmino que se produce a partir de complementar las variables en m_i ?
- A la luz del problema 26, dado el mapa de una función, interprete cómo puede construirse el mapa de su dual.
- Una función que es su propio dual se denomina una función *autodual*. De acuerdo con los problemas 25 y 26, proporcione las condiciones necesarias para que una función de n variables sea autodual.
- Utilizando el resultado del problema 28, determine cuál de las funciones cuyos mapas se muestran en la figura P29 podría ser autodual.
 - Repita para la función: $f(A, B, C, D) = \Sigma(0, 2, 3, 7, 9, 10, 14, 15)$.

	x		wx				wx			
	0	1	00	01	11	10	00	01	11	10
yz	00	1					00			
	01	1					01	1	1	1
	11		1		1	1	11		1	1
	10			1		1	10		1	1

a)

	x		wx				wx			
	0	1	00	01	11	10	00	01	11	10
yz	00	1					00	1		1
	01	1					01		1	1
	11		1		1	1	11	1		1
	10			1		1	10	1	1	

b)

	x		wx				wx			
	0	1	00	01	11	10	00	01	11	10
yz	00	1				1	00	1		1
	01				1	1	01		1	
	11	1	1				11	1		1
	10	1	1	1			10	1	1	

c)

Figura P29

30. Dada una función de conmutación g , demuestre que la siguiente función es autodual: $f = x_i g + x_i' g_d$, donde x_i es una variable de conmutación que puede o no ser una variable en g .

31. Utilizando el resultado del problema 30, construya algunas funciones autoduales utilizando cada una de las funciones g siguientes.
- $g = x + y$
 - $g = x'y$
 - $g = A + B'C$ (Use primero B como variable x , luego utilice C .)
32. Empleando un mapa lógico, muestre que una expresión mínima p de s que representa a una función autodual puede obtenerse de una expresión mínima s de p intercambiando las operaciones de suma y producto.
33. Sea $f(A, B, C, D) = f_1(A, B, C, D) + f_2(A, B, C, D)$, donde $f_1 = B'D' + A'C'D' + AB'C'$. Determine f_2 de modo tal que f será la función autodual más simple.
34. a. Suponiendo que tanto las variables como sus complementos se disponen como entradas, obtenga una realización de la siguiente expresión, sin manipularla:

$$f = xy + (w + z)(w' + z')$$

Advierta el número de niveles, como el retardo de propagación completo y el factor de carga de entrada máximo de cada compuerta. Analice su circuito y confirme que la salida es la misma que en la expresión dada.

- Determine otras cinco expresiones que representan la misma función que la de la parte a e implemente cada una. En ambas, indique el factor de carga de entrada de cada compuerta y especifique la máxima. Compare las expresiones en términos del número de niveles y del retardo de propagación, suponiendo retardos de compuerta iguales.
 - Convierta cada circuito de las partes a y b en circuitos sólo NAND. En cada caso, indique si ha aumentado el número de compuertas.
 - Construya una tabla cuyos renglones son las realizaciones en a y b . Forme columnas que correspondan al número de compuertas, número de conexiones internas, factor de carga de entrada máximo de cualquier compuerta y el retardo más largo desde una entrada hasta la salida. Compare éstas respecto a distintas realizaciones.
 - Analice cada circuito para obtener expresiones correspondientes a la salida. Confirme que cada expresión representa la misma función que en la expresión dada originalmente.
35. Repita el problema 34 para cada una de las siguiente expresiones.
- $f = C(A' + DE) + D'(B'E' + A'BE)$
 - $f = A(B' + CD') + B(AC' + D) + C'(A'B + AB'D)$
36. El número de minitérminos en una función que tiene n variables es 2^n . Demuestre que el número más grande de términos en una expresión mínima de suma de productos que representa a cualquiera de estas funciones es 2^{n-1} , la mitad del número de minitérminos.
37. Utilizando cuando mucho un inversor adicional, convierta cada circuito presentado en la figura 13 del texto en:
- Un circuito sólo NAND.
 - Un circuito sólo NOR.
38. Un comparador de 2 bits es un dispositivo lógico con dos pares de líneas de entrada x_1, x_0 y y_1, y_0 y una salida, z . Las combinaciones $X = x_1x_0$ y $Y = y_1y_0$ representan números binarios. La salida será 1 siempre que $X \geq Y$.

- Utilizando un mapa lógico, escriba una expresión de suma de productos para la salida.
- Obtenga una realización del circuito; luego conviértala a un circuito sólo NAND.
- Escriba también una expresión de producto de sumas. Indique cuál expresión tiene menos términos.
- Obtenga también una realización de circuito para este caso; conviértala en un circuito sólo NOR.
- Dibuje un diagrama de temporización que describa la salida para los casos $X < Y$, $X > Y$ y $X = Y$ (use valores apropiados de X e Y).

00
1
1

$$f = x_1g + x_1'g_d$$

39. Como se explica en la sección 7, un *comparador de magnitud* es un circuito combinatorio cuyas $2n$ entradas son los bits de dos palabras binarias de n bits, A y B . Éste cuenta con tres salidas:
- G , que es 1 cuando $A > B$
 - E , que es 1 cuando $A = B$
 - L , que es 1 cuando $A < B$
- a. Cuando $n = 2$ bits, las palabras de entradas son $A = A_1A_0$ y $B = B_1B_0$. Elabore tablas de verdad para G , E y L , considerando cada bit de cada palabra de entrada como una variable. (Este comparador es un circuito de cuatro entradas, tres salidas.)
 - b. A partir de estas tablas construya mapas lógicos para G , E y L .
 - c. Obtenga una expresión de conmutación para E como una salida en términos de G y L como entradas. Repita para G en términos de E y L . Repita para L en términos de E y G .
 - d. Obtenga una realización mínima para las tres salidas G , E y L a partir de mapas lógicos, utilizando el resultado de la parte c.
 - e. Modifique el resultado en la parte b, considerando el circuito como uno de salida múltiple. Esto es, se crean términos comunes entre las salidas al grado posible para que se reduzca el conteo total de compuertas.
40. Construya un comparador de magnitud de 4 bits a partir de dos de 2 bits. Uno de éstos comparará los bits más grandes de cada palabra y el otro los más pequeños. Las salidas G , E , y L del comparador de los bits más pequeños se convertirán en entradas adicionales para el comparador de bits más grandes. Para distinguirlas de las salidas totales G , E , y L , se denominarán G_{in} , E_{in} y L_{in} .
- a. Dibuje un diagrama de bloques, que muestre un bloque para cada comparador de dos bits, con terminales de entrada y salida apropiadas que se indiquen y marquen de manera explícita.
 - b. ¿Bajo qué condiciones el resultado de la comparación de los bits más pequeños tiene influencia en el resultado total? Explique.
 - c. Si las unidades de 2 bits van a ser módulos idénticos, especifique para el comparador de los bits más pequeños qué valores deberán tener las entradas G_{in} , E_{in} y L_{in} para que el comparador opere de modo correcto.
41. Se indican dos números binarios de 2 bits: $X = x_1x_2$ y $Y = y_1y_2$. La suma es $Z = CBA$, donde C es el acarreo.
- a. Dibuje un mapa lógico para cada uno de los dígitos de la suma: C , B , A .
 - b. A partir de los mapas, escriba expresiones para los dígitos de la suma en la forma s de p .
 - c. Reescriba estas expresiones para utilizar compuertas XOR lo más posible.
 - d. Suponga que cada compuerta tiene un retardo pequeño. ¿Hay alguna secuencia de entrada particular que ocasione que el circuito tenga un mayor retardo total que cualquier otra? Dibuje un diagrama de temporización de la secuencia.
42. Considere el circuito del problema 41 como uno de salida múltiple y obtenga una realización que reduzca el número de compuertas al mayor grado posible.
43. Un circuito tendrá cuatro entradas $wxyz$ y cuatro salidas $ABCD$. El conjunto de salida va a representar un número BCD que será el número de entrada incrementado por 1. Si la entrada es 0111, la salida será 1000. (Suponga que la entrada corresponde a 1011; ¿cuál será el número de salida?)
- a. Dibuje los mapas lógicos para cada salida.
 - b. Escriba la lista de minitérminos para cada una.
 - c. A partir de cada lista, escriba una expresión mínima s de p .
 - d. Dibuje un diagrama lógico de dos niveles que realice cada expresión.
 - e. Convierta la lista de minitérminos en b en listas de maxitérminos.
 - f. Dibuje un diagrama lógico de dos niveles para cada expresión en e .
44. a. Repita el problema 43, tratándolo como un circuito de salida múltiple. Use las relaciones entre las salidas múltiples para minimizar el número de compuertas.
- b. Convierta la realización en una sólo NAND.
 - c. Convierta la realización en una sólo NOR.

45. a. Un circuito combinatorio de cuatro entradas, una salida, va a ser un detector de números primos BCD. Siguiendo los procedimientos en el ejercicio 18 del texto, obtenga un circuito mínimo de s de p .
- b. Obtenga también un circuito mínimo p de s .
46. Dos números binarios de 2 bits $A = a_1a_0$ y $B = b_1b_0$ constituyen las entradas a un circuito lógico de cuatro entradas. La salida es un número binario de 4 bits $C = c_3c_2c_1c_0$, que va a ser el producto de A y B .
- a. Construya mapas lógicos para cada salida en términos de las cuatro variables de entrada. (Podría ayudar construir primero una tabla de verdad, aunque esto no es necesario.)
- b. A partir de los mapas, encuentre expresiones mínimas s de p para cada c_i .
- c. Construya una realización de circuito de cada salida; elabore una nota acerca de la complejidad del circuito.
47. La tabla de implicants primos para una función incompletamente especificada se muestra en la figura P47.
- a. Determine los implicants primos esenciales, si los hay; construya después una tabla reducida si es necesario.
- b. Si algunos renglones son dominados por otros, especifíquelos y elimínelos. Determine luego los implicants primos esenciales secundarios, si los hay.
- c. Se dice que una columna m_i dominará a otra columna m_j si ni, es dominada por todos los implicants primos que cubre m_j , y posiblemente también por otros implicants primos. Por ejemplo, la columna 24 domina a la columna 17. Demuestre que, si un impicante primo en la expresión mínima de s de p cubre la columna dominada, éste necesariamente cubrirá la columna dominante. En consecuencia, es posible eliminar la columna dominante.
- d. Utilice el resultado de la parte c para determinar una tabla reducida adicionalmente.
- e. Complete la determinación de una expresión mínima de s de p .

	6	9	10	12	17	20	24
P_1		✓	✓	✓			
P_2					✓	✓	✓
P_3	✓			✓			
P_4				✓			✓
P_5	✓		✓	✓			
P_6		✓			✓		✓
P_7						✓	

Figura P47

48. Un comprador de k bits se representa mediante C_k en la figura P48. Compare dos números de k bits $X_k = x_1x_2 \dots x_k$ y $Y_k = y_1y_2 \dots y_k$, con las salidas G_k y S_k de la mancha siguiente:

$$\begin{aligned}
 G_k S_k &= 10 && \text{si } X_k > Y_k \\
 &= 01 && \text{si } X_k < Y_k \\
 &= 10 && \text{si } X_k = Y_k
 \end{aligned}$$

Se va a diseñar un comparador de $(k+1)$ bits utilizando un CI de comparadores de k bits y otra unidad lógica L como se muestra en la figura. (Esto es, G_{k+1} y S_{k+1} cumplen las condiciones precedentes sobre G_k y S_k , con k reemplazada por $k+1$.)

- a. Encuentre expresiones lógicas para las salidas G_{k+1} y S_{k+1} en términos de todas las entradas a L . (Sugerencia: considere los valores posibles de G_k y S_k , y los valores resultantes de G_{k+1} y S_{k+1} .)
- b. Suponga que están disponibles los CIs que implementan L , así como las constantes 0 y 1. Demuestre que las entradas a un CI L tendrían que ser de modo tal que servirían como un comparador de números de 1 bit.
- c. Muestre un diagrama de bloques que implemente C_3 (un comparador de 3 bits) utilizando únicamente paquetes L .

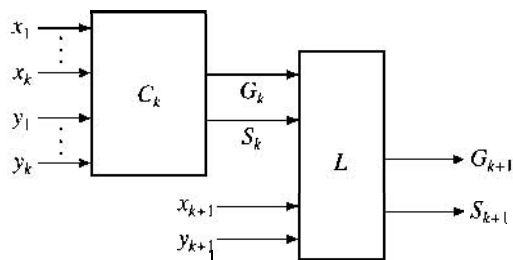


Figura P48

49. Un circuito de conmutación va a tener cuatro líneas de entrada. Se sabe que lo que reciben estas líneas serán palabras decimales codificadas en binario. La salida será 1 siempre que la palabra BCD recibida corresponda a 0, 2, 3, 5 u 8; en otro caso la salida es 0. Diseñe un circuito mínimo de dos niveles.
50. El diagrama de la figura P50 representa un circuito lógico implementado de manera parcial. Las salidas son:

$$f_1 = \Sigma(4, 9, 11, 12, 13)$$

$$f_2 = \Sigma(4, 5, 6, 7, 12, 13)$$

- Construya un mapa lógico de la función $x(A, B, C, D) = x_{\text{máx}}$, que va a tener el número máximo de minterminos mientras siga permitiendo las salidas f_1 y f_2 dadas.
- Construya mapas de y y z correspondientes a $x_{\text{máx}}$, dejando los valores sin especificar siempre que sea posible.

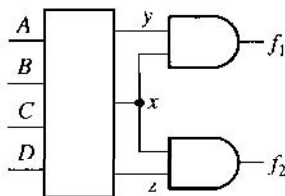


Figura P50

51. Suponga que las salidas en el circuito lógico combinacional de la figura P50 van a ser:

$$f_1 = \Sigma(0, 1, 3, 4, 5, 7, 11, 15)$$

$$f_2 = \Sigma(2, 3, 6, 7, 11, 15)$$

- De todas las funciones intermedias x posibles, encuentre una x_{min} que tenga el menor número de minterminos.
 - Muestre los mapas lógicos correspondientes para y y z , suponiendo que cada uno tiene el número máximo de minterminos.
52. La Lemon Logic Corporation ha diseñado un CI con copias múltiples de un circuito, denominado compuerta PU, con cuatro variables de entrada —A, B, C, D— y una salida marcada como PU. Esta compuerta implementa la función.

$$PU(A, B, C, D) = BC(A + D)$$

Los ingenieros de diseño en Lemon están investigando la posible implementación de funciones de conmutación, utilizando lógica PU-OR. Para ayudarles, diseñe un circuito que implementa la siguiente función, utilizando exclusivamente tres compuertas PU y una compuerta OR. (Suponga que tanto la variable como su complemento se disponen como entradas para PU.)

$$f(x_1, x_2, x_3, x_4) = \Sigma(0, 1, 6, 9, 10, 11, 14, 15)$$

53. Lleve a cabo un proyecto de investigación para descubrir algunas propiedades de la función OR exclusiva de tres variables: $f = x \oplus y \oplus z$.

- a. Encuentre una expresión booleana para f en términos de x , y y z .
 - b. A partir de ésta, escriba f como una lista de minterminos.
 - c. Escriba los términos en la lista de minterminos como números binarios. A partir de un examen de estos números, extraiga dos conclusiones generales acerca de los minterminos de $x \oplus y \oplus z$.
54. Repita el problema 53 para la OR exclusiva de cuatro variables, $f = w \oplus x \oplus y \oplus z$.
55. El objetivo de este problema es diseñar un circuito que utilice un sumador completo sencillo más algún otro elemento para sumar dos números binarios de n bits, un bit a la vez. Describa lo que sería ese "algún otro elemento", y ejemplifique con los números 1101 y 0110.
56. Un multiplicador tiene dos pares de líneas de entrada, a_1a_0 y b_1b_0 . Las entradas de dos dígitos en estas líneas representan palabras binarias de dos dígitos. El multiplicador cuenta con cuatro líneas de salida; la palabra $p_3p_2p_1p_0$ que aparece en estas líneas representa el producto de esos números.
- a. Escriba expresiones lógicas para cada uno de los dígitos de producto p_i .
 - b. Obtenga una realización para cada dígito.
 - c. En la medida posible, utilice compuertas comunes compartidas por más de una salida.
57. Un circuito se construye con dos compuertas XOR de la manera siguiente. La primera compuerta tiene dos entradas externas, x_1 y x_2 . Las entradas a la segunda XOR son la entrada externa x_3 y la salida de la primera XOR. Recuerde la relación de las entradas necesarias para que una compuerta XOR produzca una salida 1.
- a. Dibuje el circuito resultante y especifique las relaciones de las tres entradas necesarias para producir una salida 1.
 - b. Amplíe el circuito agregando una cuarta compuerta XOR, una de cuyas entradas es externa y la otra es la correspondiente a la salida del circuito previo. Especifique también en este caso la relación entre las entradas necesarias para producir una salida 1.

Las XOR adicionales pueden sumarse de la misma manera, dando lugar a la misma relación entre las entradas necesarias para producir una salida 1. Esta estructura se conoce como *cadena tipo margarita (daisy chain)*, y sirve como un detector de paridad impar. Verifique esto para los casos en a y b .

58. Algunas veces un circuito tendrá varias salidas, dependientes todas del mismo conjunto de variables de entrada. El procedimiento que se usó en este capítulo puede aplicarse para implementar cada una de las funciones de salida independientemente de las otras. Sin embargo, a veces es posible utilizar implicantes primos que son comunes entre dos o más de las funciones de salida. En esos casos la misma compuerta se usa en las trayectorias desde las entradas hasta dos o más de las salidas. En realidad, podría resultar rentable elegir implicantes que son comunes entre varias de las salidas incluso si éstos no son implicantes primos. El compromiso aquí es aceptar entradas adicionales a ciertas compuertas con el beneficio de un número total de compuertas inferior.

Los siguientes conjuntos de funciones son salidas que dependen de los mismos conjuntos de entradas. En cada caso, considere dos implementaciones diferentes:

Las sumas de productos mínimas implementan de manera independiente cada función.

Las implementaciones de suma de productos que utilizan compuertas comunes entre las tres funciones, o entre pares de ellas.

Compare los números de compuertas, entradas y CIs SSI necesarios para implementar de manera independiente cada una de las funciones.

- a. $f_1 = \Sigma(0, 1, 8, 9, 14, 15)$
 $f_2 = \Sigma(6, 7, 12, 13, 14, 15)$
 $f_3 = \Sigma(8, 9, 12, 13, 14, 15)$
- b. $f_1 = \Sigma(0, 1, 4, 8, 10, 15)$
 $f_2 = \Sigma(0, 1, 5, 6, 7)$
 $f_3 = \Sigma(1, 6, 9, 13, 14)$
- c. $f_1 = \Sigma(1, 3, 4, 5, 7, 9, 13, 18, 19, 20, 21, 26, 27)$
 $f_2 = \Sigma(4, 5, 6, 9, 12, 13, 14, 20, 21, 22, 23, 18, 29)$
 $f_3 = \Sigma(6, 7, 9, 11, 12, 13, 14, 15, 18, 19, 20, 21, 22, 23)$

- d. $f_1 = \Sigma(0, 4, 5, 11, 12) + \Sigma d(6, 10)$
 $f_2 = \Sigma(4, 8, 10, 12) + \Sigma d(0, 14)$
 $f_3 = \Sigma(4, 10, 12, 14) + \Sigma d(6, 7, 8)$
- e. $f_1 = \Sigma(1, 5, 7, 4, 20, 22, 29, 30) + \Sigma d(3, 4, 13, 21, 25, 28)$
 $f_2 = \Sigma(2, 6, 7, 14, 22, 25, 28, 30, 31) + \Sigma d(1, 12, 23, 27, 29)$
 $f_3 = \Sigma(6, 8, 12, 14, 22, 24, 26, 30) + \Sigma d(4, 10, 17, 18, 28)$

Capítulo 4

Diseño lógico combinatorio

En los capítulos anteriores se establecieron los fundamentos para el diseño de los circuitos lógicos digitales. Los elementos del álgebra booleana (álgebra de conmutación de dos elementos) y la forma de representar con ella las operaciones de manera esquemática mediante compuertas (dispositivos primitivos) se analizaron en el capítulo 2. Cómo manipular y representar las expresiones de conmutación de diferentes maneras constituyó el tema del capítulo 3, el cual mostró diversas formas de efectuar dichas representaciones en una diversidad de circuitos que utilizan compuertas primitivas.

Con todas esas herramientas disponibles para cumplir con el objetivo presente, en este capítulo nos interesa el diseño de circuitos lógicos más complejos. Los circuitos en los que todas las salidas en un tiempo determinado dependen sólo de las entradas en ese tiempo reciben el nombre de circuitos lógicos combinatorios. Los procedimientos de diseño se ilustrarán con clases importantes de circuitos que ahora son universales en los sistemas digitales.

El método aplicado consiste en examinar las tareas que pretende que efectúe un circuito lógico combinatorio y en identificar después uno o más circuitos que puedan ejecutar la tarea. Es probable que un circuito tenga algunas ventajas específicas sobre otros, aunque también puede incluir ciertas deficiencias. A menudo es factible mejorar un factor, pero sólo a expensas de otros. Entre los factores importantes se encuentra la velocidad de operación, la complejidad o el costo del hardware, la disipación de potencia y la disponibilidad de las unidades prefabricadas. Consideraremos varias operaciones diferentes que resultan útiles en distintos contextos y mostraremos cómo diseñar los circuitos apropiados para efectuar estas operaciones.

1 SUMADORES BINARIOS

Una de las operaciones más importantes que llevó a cabo una computadora digital es la suma de dos números binarios¹. Una medida útil del desempeño es la velocidad. Desde luego, ésta se incrementa utilizando familias lógicas de compuertas que la favorecen a costa de otras medidas, como el consumo de potencia (utilizando la familia Schottky avanzada, por ejemplo, en vez de la Schottky de baja potencia). Sin embargo, para el diseñador lógico, la pregunta importante consiste en cómo diseñar un sumador para incrementar la velocidad, prescindiendo del tipo de compuerta utilizada. Es factible que esa velocidad incrementada se alcance a expensas de una mayor

¹ Como se explicó en el capítulo 1, la resta de dos números se incluye en el significado de la suma, ya que la resta se realiza primero efectuando alguna operación en el sustraendo y sumando después el resultado (Cuál de las operaciones se efectúa primero depende del tipo de computadora, ya sea invirtiendo el sustraendo o tomando su complemento a dos, como se indica en el capítulo 1.)

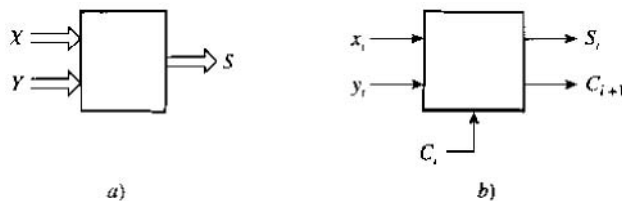


Figura 1. Suma binaria a) Sumador general, b) Sumador completo de dos palabras de un bit.

complejidad del circuito. Esta es, habría varios diseños, cada uno caracterizado por cierta velocidad y cierta complejidad de circuito. Es necesario efectuar una valoración en cuanto a los compromisos aceptables entre ellas.

En la figura 1a se muestra un diagrama simbólico que representa un sumador binario. Cada flecha abierta representa múltiples variables; en este caso las entradas son dos números binarios. Si cada número tiene n dígitos, entonces cada línea indicada representa realmente n líneas. La suma de dos números de n bits es un número de $(n + 1)$ bits. De tal manera, S (suma) representa $n + 1$ líneas de salida. Si el circuito se diseñara por medio de los métodos del capítulo 3, requeriría un circuito con $n + 1$ funciones de salida cada una dependería de $2n$ variables. La tabla de verdad para cada una de las cuales de las funciones de salida tendría 2^{2n} renglones. Puesto que n podría estar fácilmente en la gama de 20-40, es obvio que se necesita un método diferente.

Sumador completo

Otro método para sumar dos números de n bits consiste en utilizar circuitos separados para cada par correspondiente de bits. Un circuito de estas características aceptaría los 2 bits que se van a sumar, junto con el acarreo resultante de la suma de los bits menos significativos. Se producirían como salidas un hit de la suma y un hit del acarreo de salida del bit más significativo. Un circuito como éste se conoce como *sumador completo*. En la figura 1b se presenta un diagrama esquemático. Los 2 bits por sumar son x_i e y_i , y el acarreo de entrada es C_i . Las salidas son la suma S_i y el acarreo de salida C_{i+1} . La tabla de verdad para el sumador completo y los mapas lógicos para las dos salidas se ilustran en la figura 2.

Las expresiones mínimas de suma de productos para las dos salidas obtenidas de los mapas son:

$$S_i = x_i' y_i C_i' + x_i y_i' C_i' + x_i' y_i C_i + x_i y_i C_i \quad (1a)$$

$$\begin{aligned} C_{i+1} &= x_i y_i + x_i C_i + y_i C_i \\ &= x_i y_i + C_i (x_i + y_i) \end{aligned} \quad (1b)$$

(Asegúrese de verificar esto.) Cada minitérmino en el mapa de S_i constituye un implicante primo. En consecuencia, una expresión de suma de productos requeriría cuatro compuertas AND de 3 entradas y una compuerta OR de 4 entradas. El acarreo requerirá tres compuertas AND y una compuerta OR. Si suponemos que cada compuerta tiene el mismo retardo de propagación t_p , entonces una implementación de dos niveles tendrá un retardo de propagación de $2t_p$.

En el mapa del acarreo, el minitérmino m_7 se cubre por medio de cada uno de los tres implicantes primos. Esto es excesivo; puesto que m_7 se cubre mediante el implicante primo $x_i y_i$, no hay necesidad de cubrirlo otra vez utilizándolo para formar implicantes primos con m_5 y m_6 . Si existe algún beneficio al respecto, podríamos utilizar los últimos dos minitérminos como implicantes sin formar implicantes primos con m_5 . La expresión resultante para C_{i+1} se vuelve

$$C_{i+1} = x_i y_i + C_i (x_i' y_i + x_i y_i') = x_i y_i + C_i (x_i \oplus y_i) \quad (2)$$

C_i	X_i	Y_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

a)

	x	
	0	1
00		1
01	1	
11		1
10	1	

b)

	x	
	0	1
00		
01		1
11	1	1
10		1

c)

Figura 2. Tabla de verdad y mapas lógicos del sumador completo. a) Tabla de verdad; b) mapa S_i ; c) mapa C_{i+1} .

(Confirme este resultado.) Ya tenemos una expresión para S_i en 1a, pero está en forma canónica de suma de productos. Resultaría útil buscar una forma alternativa para una implementación más útil.

Ejercicio 1. Conforme, con álgebra de conmutación, confirme que la expresión para la suma en 1a puede convertirse en

$$S_i = x_i \oplus y_i \oplus C_i \quad (3) \bullet$$

Empleando las expresiones para S_i y C_{i+1} , que contienen operaciones XOR, confirme que si es posible obtener la implementación del sumador completo que se muestra en la figura 3a. Advierta que el circuito consta de dos combinaciones XOR y AND, y una compuerta OR adicional. La figura 3b muestra el circuito dentro de cada caja punteada, que recibe el nombre de *semisumador*. Sus únicas entradas son los 2 bits que se van a sumar, sin un acarreo de entrada. Las dos salidas son: 1) la suma de los 2 bits y 2) el acarreo de salida.

Suponiendo que una compuerta XOR (realizada mediante un circuito de dos niveles) tenga un retardo de propagación de $2t_p$, el sumador completo en la figura 3a tiene un retardo de propagación de $4t_p$, tanto para la suma como para el acarreo (verifique estas afirmaciones).

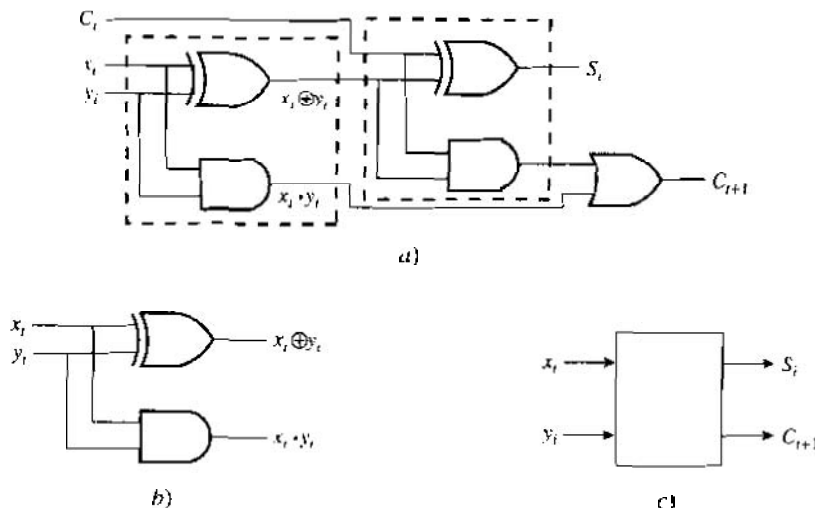


Figura 3. Sumador completo implementado con sumadores medios. a) Sumador completo. b) Semisumador. c) Diagrama esquemático del semisumador.

En la siguiente sección observaremos que la velocidad total en la suma de dos números binarios de n bits depende principalmente de la velocidad con la que se propaga el acarreo desde el bit menos significativo hasta el más significativo. Por consiguiente, reducir el retardo experimentado por el acarreo de un sumador completo es una mejora importante. Esto es un incentivo en la búsqueda de otras realizaciones del sumador completo. En algunos de los casos en el problema 1 al final del capítulo, se proponen realizaciones adicionales del sumador completo en las que el retardo de propagación para el acarreo es $2t_p$ en lugar de $4t_p$. De aquí en adelante, para un sumador completo, supondremos que el retardo de propagación del acarreo corresponde a $2t_p$.

Sumador de acarreo propagado

El problema de sumar dos números binarios multidígito se formula de la siguiente manera. Se dispone de dos números binarios de n bits, con todos los dígitos en paralelo. La suma se lleva a cabo realizando un sumador completo para sumar cada par correspondiente de dígitos, uno a partir de cada número. Los sumadores completos se conectan en *tándem* de manera que el acarreo de salida de una etapa viene a ser el acarreo de entrada de la siguiente, como se ilustra en la figura 4 para el caso de números de cuatro dígitos. Así, el acarreo se propaga a lo largo de cada etapa. En la suma binaria, el acarreo en la primera etapa (menos significativa) es 0. El acarreo final (el acarreo desbordado) se convierte en el bit más significativo de la suma de $(n + 1)$ bits.

Puesto que el acarreo de cada sumador completo tiene un retardo de propagación de $2t_p$, el retardo total al efectuar la suma de dos números de n bits es $2nt_p$. No todos los pares de los números de n bits experimentarán este gran retraso. Considere como ejemplo los siguientes dos números;

101010

010101

Suponiendo que el acarreo de entrada a la primera etapa sea cero, al efectuar la suma no se genera ningún acarreo en ninguna etapa. En consecuencia, no habrá acarreo propagado y, por ello, ningún retardo de propagación a lo largo de la cadena de acarreo.

Sin embargo, para manejar el caso general, debe anticiparse el peor caso; ningún número nuevo debe presentarse para la suma antes que el retardo total representado por el peor caso. La máxima velocidad de suma, por tanto, está limitada por el peor caso del retardo de la propagación del acarreo.

Sumador de acarreo anticipado

Al considerar la suma de dos números binarios de n dígitos, nos pasó la idea de un solo circuito combinatorio de todas esas entradas. Por ello consideramos el uso propagada de un circuito más simple, un sumador completo, con el menor número posible de entradas. Sin embargo, lo que se gana en simplicidad del circuito con este método se pierde en velocidad. Puesto que a ésta la limita el retardo en la función de acarreo, parte de la velocidad perdida podría volverse a ganar si pudiéramos diseñar un circuito —justo para el acarreo— con más de 2 entradas, pero no tantas como $2n$. Suponga que varias etapas del sumador completo se tratan como una unidad. Las entradas a la unidad son el acarreo de entrada a la unidad, así como los dígitos de entrada a todos los

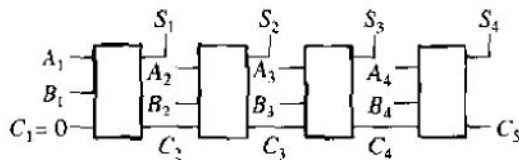


Figura 4. Sumador de acarreo propagado de 4 bits.

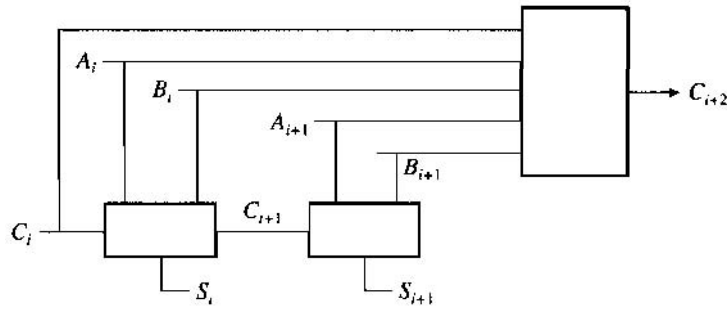


Figura 5. Diagrama esquemático del circuito de acarreo anticipado.

sumadores completos en dicha unidad. Entonces quizá el **acarreo de salida** podría obtenerse **más rápido que el acarreo propagado** a través del mismo número de sumadores completos.

Estos conceptos se **ilustran en la figura 5 con una unidad que apenas consta de dos sumadores completos y un circuito de acarreo anticipado**. Los cuatro dígitos que se van a sumar, así como el acarreo de **entrada** C_i , se presentan en forma **simultánea**. Es posible obtener una expresión para el acarreo de **salida**, C_{i+2} , de la **unidad utilizando la expresión para el acarreo del sumador completo en 2)**.

Por razones que se aclararán **más adelante**, vamos a asignar **nombres** a los dos términos en la expresión de acarreo en 2), cambiando **los nombres de las variables en A y B** de x y y de acuerdo con la figura 5. Definimos el acarreo **generado** G_i y el **acarreo propagado** P_i para el i -ésimo sumador completo de la manera siguiente:

$$G_i = A_i B_i \quad (4a)$$

$$P_i = A_i \oplus B_i \quad (4b)$$

Al insertar éstos en la expresión para el acarreo de **salida en 2)**, obtenemos

$$C_{i+1} = A_i B_i + C_i (A_i \oplus B_i) = G_i + P_i C_i \quad (5)$$

Un acarreo se **generará** en el i -ésimo sumador completo (esto es, $G_i = 1$) si A_i y B_i son ambos iguales a 1. Pero **si sólo uno de ellos es 1, no se generará un acarreo de salida**. En ese caso, **sin embargo**, P_i será 1 (confirme esto). En consecuencia, el acarreo de **salida será** $C_{i+1} = C_i$. Decimos que el acarreo **se propagará hacia adelante**.

La expresión para el acarreo de **salida en 5)** puede actualizarse cambiando el índice i a $i + 1$:

$$\begin{aligned} C_{i+2} &= C_{i+1} + P_{i+1} C_{i+1} = G_{i+1} + P_{i+1} (G_i + P_i C_i) \\ &= C_{i+1} + P_{i+1} C_i + P_{i+1} P_i C_i \end{aligned} \quad (6)$$

Es posible interpretar la última expresión de la siguiente forma. Un acarreo aparecerá en la **salida de la unidad de acuerdo con tres circunstancias**:

- En la última etapa se genera: $G_{i+1} = 1$.
- En la primera etapa se genera, $G_i = 1$, y se propaga **hacia adelante**: $P_{i+1} = 1$.
- El acarreo de entrada C_i se propaga a través de **ambas etapas**: $P_i = P_{i+1} = 1$.

Evidentemente, este resultado **puede extenderse** a cualquier número de etapas, aunque el circuito se volverá **progresivamente más complicado**.

Ejercicio 2. Extienda el resultado previo una **etapa más** y escriba la expresión para C_{i+3} . Escriba después las formas en las cuales **este** acarreo de salida puede ser 1. Confirme su resultado utilizando el resultado general que se da a continuación. •

Extendiendo el diseño a j etapas, la expresión en 6) se transforma en

$$G_{i+j+1} = G_{i+j} + P_{i+j}G_{i+j-1} + P_{i+j}P_{i+j-1}G_{i+j-2} + \dots + (P_{i+j}P_{i+j-1} \dots P_i)C_i \quad (7)$$

Esta expresión se ve complicada, pero es fácil de interpretar. Puesto que el acarreo de salida $C_{i+j+1} = 1$ si cualquiera de los términos aditivos a la derecha es 1, el acarreo de salida de la unidad será 1 para diversas posibilidades. Se genera ya sea en la última etapa (i -ésima) de la unidad o en la etapa anterior, y se propaga a través de las etapas sucesivas, o el acarreo de entrada de la unidad se propaga por todas las etapas hacia la salida.

Cuanto mayor sea el número de etapas de sumador completo incluidas en una unidad, tanto mayor será la mejora en la velocidad —aunque también es grande la complejidad del circuito de acarreo anticipado. Existe un compromiso obvio entre los dos. Considere una unidad de cuatro etapas, la cual va a sumar dos palabras de 4 bits A y B . Tal etapa puede considerarse como si tuviera un circuito de suma (S) y un circuito de acarreo separado (C). El circuito de suma de cada etapa tiene como entradas el acarreo de la etapa anterior y los bits correspondientes de las palabras A y B . Todas las entradas a la red de acarreo de cada etapa son todos los bits de las palabras A y B hasta esa etapa y el acarreo no sólo de la etapa precedente, sino de la entrada a la unidad completa. Así, si la primera etapa es la i , las entradas al circuito de acarreo de la etapa $i + 2$ son: $A_i, A_{i+1}, A_{i+2}, B_i, B_{i+1}, B_{i+2}$, y C_i .

Ejercicio 3. Dibuje un diagrama esquemático para una unidad de tres etapas utilizando rectángulos para representar los circuitos de suma y acarreo de cada etapa (suponga que la primera etapa es designada con 1 en vez de la i general).

La figura 6 muestra una realización de circuito de la red de acarreo de la última etapa en una unidad de cuatro etapas. Salvo por C_i , el acarreo de entrada de la unidad, las otras entradas a las compuertas AND son acarreos generados y acarreos propagados desde las diversas etapas de la unidad. Estos acarreos generados y propagados los producen los circuitos de semisumador de la figura 7.

La figura 8 ilustra un diagrama de semibloques del sumador de acarreo anticipado de cuatro etapas. (Note que se supone que las terminales que llevan la misma etiqueta en diferentes subcircuitos están conectadas.) Puesto que cada acarreo propagado P_{i+j} es la salida de una com-

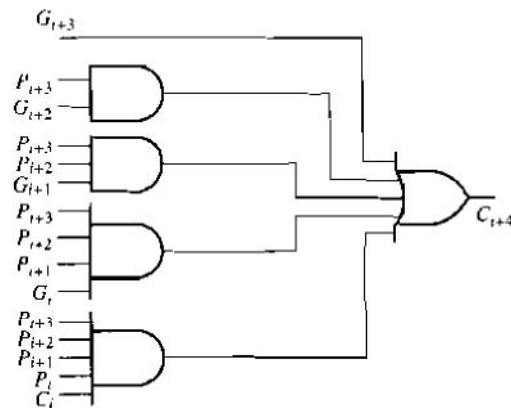


Figura 6. Circuito de acarreo anticipado de cuatro etapas.

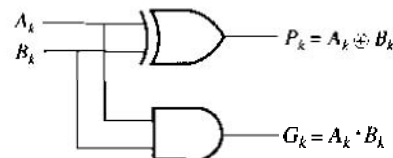


Figura 7. Semisumador para acarreos generados y propagados.

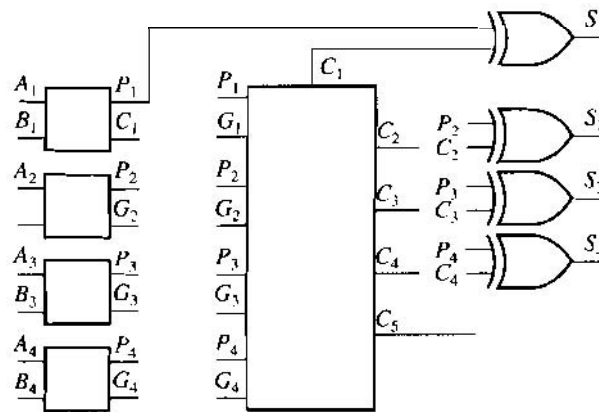


Figura 8. Diagrama esquemático del sumador de acarreo anticipado de 4 bits.

puerta XOR, el retardo total de la propagación del circuito de acarreo que tiene el diseño de la figura 7 corresponde a $4t_p$. Sin embargo, todos los acarreos generados y propagados, G_{i+j} y P_{i+j} , de todas las unidades quedan disponibles dentro de $2t_p$, después de que las dos palabras se presentan primero para la suma, como lo indica la figura 6. Por tanto, en todas las unidades de acarreo anticipado además de la primera, el retardo de propagación de la red de acarreo es únicamente $2t_p$.

Ejercicio 4. Suponga que un sumador de acarreo anticipado va a tener k unidades de 4 bits para efectuar la suma de dos palabras de $4k$ bits. De la explicación anterior, a partir del diagrama de la figura 8 que implementa cada uno, y de la consideración de la primera y última unidades, determine el retardo de propagación de este sumador en términos de t_p el retardo de propagación a través de una compuerta. (No consulte la respuesta hasta que realice el ejercicio.)

Respuesta²

Si un sumador tiene ocho unidades de 4 bits, el retardo de propagación a través de un sumador de acarreo anticipado será $20t_p$. El sumador de acarreo correspondiente propagado tendrá un retardo de propagación igual a $4 \times 8 \times 2t_p = 64t_p$. De esta manera, el sumador de acarreo anticipado tendrá una ventaja de 320% en velocidad sobre el de acarreo propagado. Sin embargo, no todo es ganancia fácil: la ventaja de velocidad se ha pagado a costa de hardware adicional.

Ejercicio 5. A partir del conteo del número de compuertas en cada implementación, estime la desventaja de hardware en porcentaje del sumador de acarreo anticipado en comparación con el sumador de acarreo propagado. Compare la desventaja con la ventaja de velocidad de 320 por ciento.

Los circuitos que se describen aquí se obtienen en CI monolíticos. Un sumador completa sencillo, por ejemplo, se dispone como una unidad. Un sumador de acarreo propagado, como se ilustra en la figura 4, y uno de acarreo anticipado para palabras de 4 bits, como se indica en la Figura 8, se consiguen como CI MSI.

² La suma de los retardos a través de a) el circuito de acarreo de cada unidad ($2t_p$ cada uno), b) el circuito de suma de la última unidad ($2t_p$) puesto que éste depende de tener el acarreo de la última unidad y c) el retardo adicional al obtener el acarreo de la primera unidad. Retardo total = $(k + 1 + 1)2t_p = (2k + 4)t_p$.

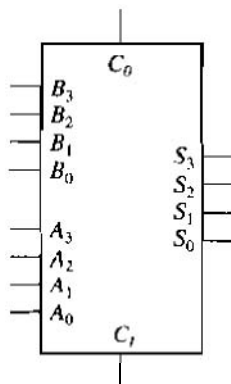


Figura 9. Sumador de alta velocidad, palabras de 4 bits.

Externamente, un CI consistente en un **sumador** de acarreo propagado de palabras de 4 bits se **vería** igual que un paquete que consta de un sumador de acarreo **anticipado** de palabras de 4 bits. El diagrama de bloques de la figura 9 ilustra un CI de este tipo.

Hay **nueve** entradas: el **acarreo** de entrada y cuatro entradas por palabra.

Son cinco **las salidas**: el **acarreo** de salida y los 4 bits de **la suma**. (El **acarreo** de salida se vuelve el bit más significativo de la suma si el circuito se usa **justo** para suñar palabras de 4 bits, y **no** como **parte** de un sumador de palabras más largas.)

Restador binario

En el capítulo 1 se estudiaron dos representaciones de números binarios con signo: **complemento a uno** y **complemento a dos**. Recuerde que **cuando** los números **se** representan en **una** de **las** formas de complemento, el único tratamiento **especial** que se **necesita** en la suma de un número negativo con otro número positivo o negativo se encuentra en el acarreo de salida final. De tal manera, los sumadores que se estudiaron en la **sección** anterior resultan adecuados para la suma de números representados en **una** de estas formas si se usa alguna **circuitería** adicional para procesar el acarreo de salida final. **Adetiás**, la resta binaria puede efectuarse utilizando los **mismos** circuitos sumadores al negar **el** sustraendo.

Sumador y restador en complemento a dos

Recuerde del capítulo 1 que cuando la suma de 2 números binarios con complemento a dos produce un acarreo final, éste puede ignorarse.

Sin embargo, es necesario detectar **el** **desbordamiento** que tal vez ocurra **cuando** el resultado de la suma está fuera de rango.³ En el capítulo 1 se concluyó que un desbordamiento aritmético podría detectarse si resultan **diferentes** el acarreo de entrada y el **acarreo de salida de la posición del bit más significativo**. Así, es posible detectar el **desbordamiento** con una compuerta OR exclusiva adicional. El sumador de complemento a dos no es **muy** diferente del sumador binario para números **sin** signo.

¿**Qué** ocurre con la resta? Ya sugerimos que ésta debe efectuarse complementando el **sustraendo** y sumando. **Por** ello la tarea consiste en diseñar un circuito cuya salida es el complemento a dos de la entrada, y en utilizar su salida como **una** entrada para un **sumador**. Un circuito de estas características se diseña sin dificultad, pero ¿**por** qué un sistema debe contener parte del **hardware** dedicado a la suma y otra **parte** **destinado** a la resta? Si la única diferencia entre estos dos circuitos es un circuito que calcula el complemento a dos, entonces ¿**por** qué no diseñar un circuito en **el** que cualquier **suma** o resta pueda **elegirse** con una entrada adicional? Cuando es-

³ El intervalo de números binarios que tienen n dígitos binarios representados en forma de complemento a dos es $-2^{n-1} \leq m \leq 2^{n-1} - 1$.

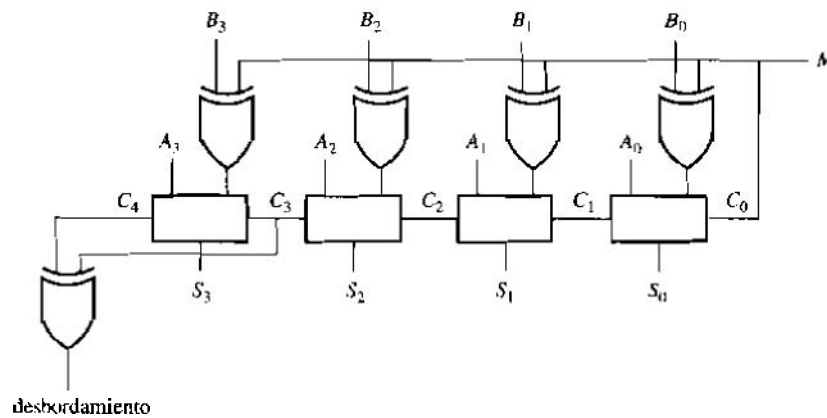


Figura 10. Sumador/restador de complemento a dos con detección de desbordamiento.

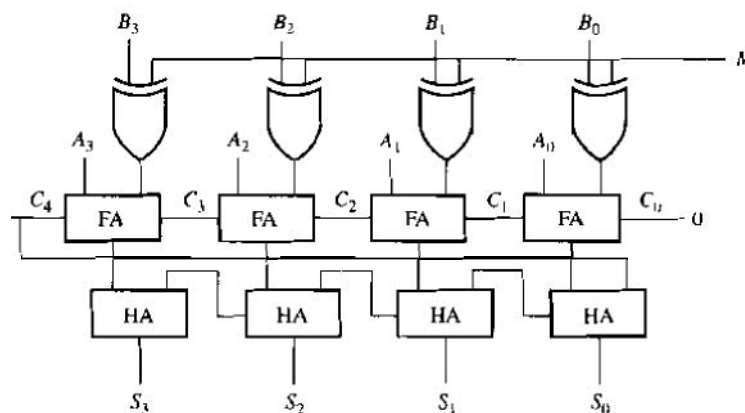


Figura 11. Sumador/restador de complemento a uno

ta última es, digamos, 0, el circuito efectúa la suma, y cuando la entrada corresponde a 1, ejecuta la resta. Parece sencillo; una representación del circuito puede obtenerse utilizando las técnicas del capítulo 3, aunque existe una solución elegante que describimos a continuación.

Examine la tabla de verdad de la operación OR exclusiva y advierta que es posible considerarla como un inversor condicional. Si una entrada es 0, entonces la salida es idéntica a la segunda entrada. Si una entrada es 1, entonces la salida corresponde al complemento de la segunda entrada. Esto es conveniente para producir el complemento de una entrada para nuestro circuito sumador/restador cuando queremos efectuar la resta. Sin embargo, para calcular el complemento a dos de 1 número binario tenemos que sumar 1. ¿Alguna idea acerca de cómo efectuar lo anterior sin compuertas adicionales? (Piense en ello antes de continuar.)

El sumador completo para el bit menos significativo tiene una señal de entrada de acarreo que puede utilizarse para sumar el 1 requerido.

El diseño de nuestro circuito sumador/restador de complemento a dos está completo; una versión para sumar números de 4 bits se muestra en la figura 10. Si la señal de control M es 0, entonces el circuito efectúa $A + B$; sin embargo, si M es 1, el circuito realiza $A - B$.

Sumador y restador de complemento a uno

Para efectuar la resta en complemento a uno es posible utilizar el circuito OR exclusivo que se empleó en el sumador/restador de complemento a dos. La única diferencia es que no deseamos

inyectar un acarreo en el bit menos significativo. La suma de complemento a uno requiere añadir 1 a la suma cuando ocurre un acarreo de salida a partir de la posición del bit más significativo. Esto puede conseguirse utilizando semisumadores múltiples como se muestra en la figura 11. La detección de desbordamiento para la suma de complemento a uno se deja como problema al lector.

La suma de complemento a dos es el método más común que se implementa en las computadoras modernas debido a su reducida complejidad de circuito en comparación con el complemento a uno.

Éste es el punto hasta donde llegaremos en la suma de palabras multibit; otros circuitos sumadores se dejan para los problemas de fin de capítulo.

2 MULTIPLEXORES

En las comunicaciones, control y sistemas de computadora pueden ejecutarse muchas operaciones mediante circuitos lógicos combinatorios. Cuando un circuito se ha diseñado para efectuar alguna tarea en una aplicación, a menudo también encuentra empleo en diferentes aplicaciones. De este modo, adquiere diferentes nombres a partir de sus diversos usos. En ésta y en las secciones siguientes describiremos varias de estos circuitos y sus empleos. Explicaremos sus principios de operación, especificando sus implementaciones MSI o LSI.

Una operación común se ilustra en la figura 12. Los datos que se generan en una localidad se van a usar en otra. Se necesita un método para transmitirlos de una localidad a otra a través de algún canal de comunicaciones.

Los datos están disponibles, en paralelo, en muchas líneas diferentes, pero deben transmitirse por un solo enlace de comunicación. Se necesita un mecanismo para elegir en forma secuencial cada una de las líneas de datos de manera que los datos que la línea seleccionada porta, puedan transmitirse en ese momento. Este proceso recibe el nombre de *multiplexado*. Un ejemplo es el multiplexado de conversaciones en el sistema telefónico. Varias conversaciones en la línea telefónica se conmutan de manera alternada muchas veces por segundo. Debido a la naturaleza del sistema auditivo humana, los oyentes no pueden detectar lo que escuchan si hay interrupciones y las conversaciones de otras personas se mezclan con las propias en el proceso de transmisión.

En el otro extremo del enlace de comunicación está un dispositivo necesario que deshará el multiplexado: un *desmultiplexor*. Éste debe aceptar los datos en serie entrantes y dirigirlos en paralelo a una de muchas líneas de salida. Los trozos intermezclados de las conversaciones telefónicas, por ejemplo, deben separarse y enviarse a los oyentes correctos.

Un multiplexor digital es un circuito con 2^n líneas de entrada de datos y una línea de salida; también debe tener una manera de determinar la línea de entrada de datos específica que se va a seleccionar en cualquier momento. Esto se efectúa con otras n líneas de entrada, denominadas *entradas de selección*, cuya función es elegir una de las 2^n entradas de datos para la conexión con

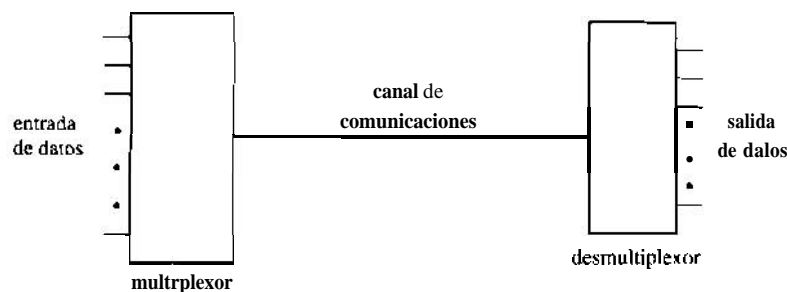


Figura 12. Un problema de comunicación de datos.

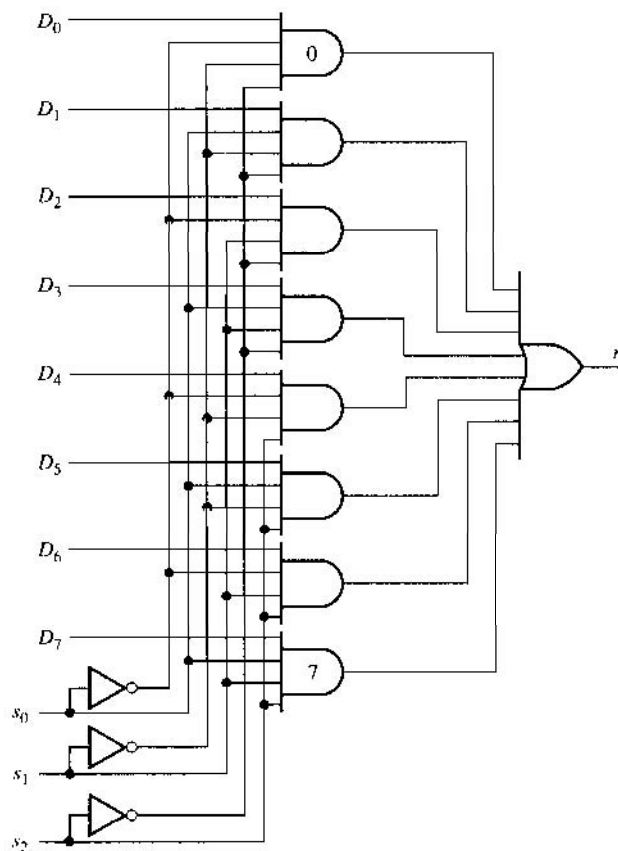


Figura 13. Multiplexor con ocho entradas de datos.

la salida. Un circuito para $n = 3$ se muestra en la figura 13. Las n líneas de selección tienen $2^n = 8$ combinaciones de valores que constituyen **números de selección** binarios.

Ejercicio 6. Escriba expresiones para cada una de las salidas de compuertas AND en términos de las entradas s_i y D_j , confirmando que el multiplicador de D_k es el equivalente binario de k . •

Cuando las entradas de selección tienen la combinación $s_2s_1s_0 = 011$, por ejemplo, las salidas de todas las compuertas AND serán 0, **excepto** aquella a la cual se conectan la línea de datos D_3 . Las otras entradas a la compuerta AND diferentes de D_3 serán 1. Por consiguiente, D_3 aparece a la salida del circuito. **De este modo, las entradas de selección** cuya combinación binaria corresponde al decimal 3 han **elegido la entrada** de datos D_3 para transmitirse a la salida.

Es posible obtener CI MSI estándar como multiplexores. La figura 14a muestra el circuito para un CI que contiene dos multiplexores independientes para $n = 2$. Las consideraciones prácticas no incluidas en la figura 13 explican algunos de los rasgos de este circuito. La entrada **habilitadora** E , por ejemplo, se utiliza **para controlar el periodo de tiempo** en el que el multiplexor está operando. **Así, cuando el valor de E es 1, la salida será 0** sin importar los valores de las entradas de selección. El circuito estará operando únicamente cuando la entrada habilitadora correspondiente es 0. (En otros circuitos, la señal habilitadora **no está invertida**; en **tales casos**, el circuito opera cuando $E = 1$, exactamente lo opuesto al caso mostrado en la figura 14a.)

Además, advierta en la figura que tanto las **señales de selección** como **sus complementos** son entradas para las compuertas AND. Las mismas entradas de señal se obtienen después de dos

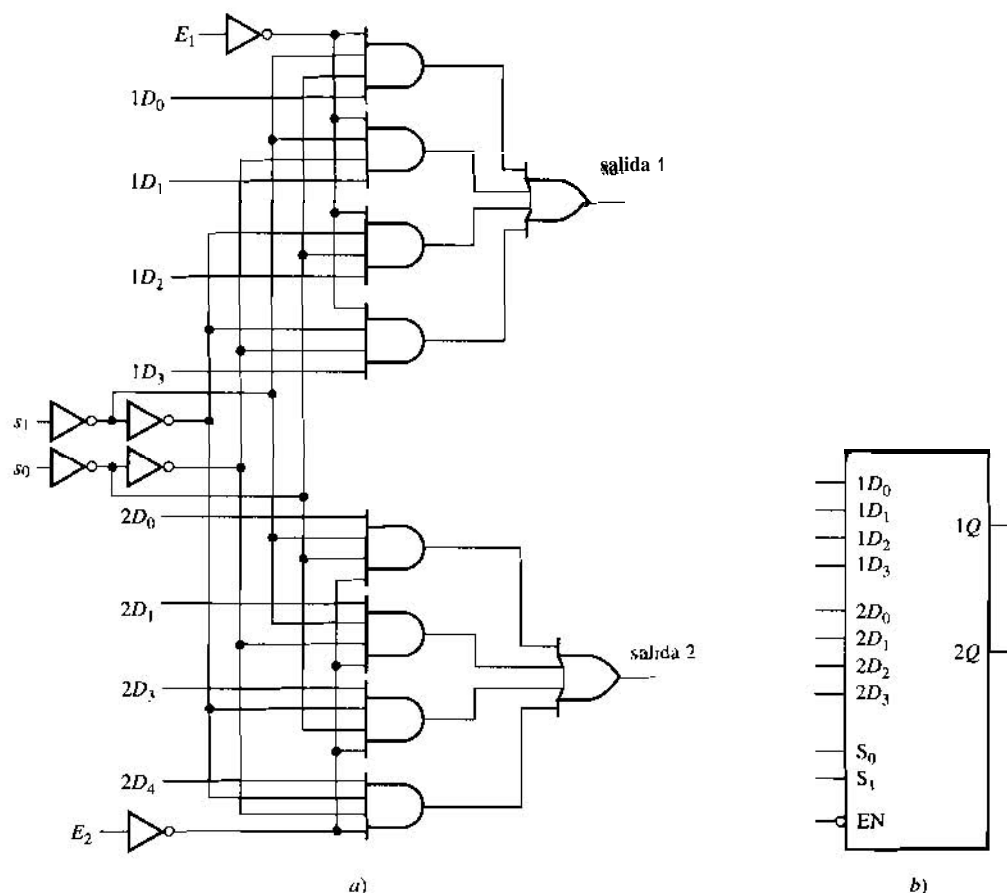


Figura 14. a) Multiplexor dual de cuatro entradas con habilitador. b) Multiplexor dual de cuatro entradas con habilitador sencillo.

inversiones, lo cual resulta especialmente útil si n es grande. En esta forma, el circuito que produce las entradas de selección tiene como carga sólo una compuerta simple (el inversor) en vez de varias compuertas AND. En la figura 14a las entradas de selección son comunes a ambos multiplexores, aunque cada uno tiene su propio habilitador. En otros diseños, el habilitador también puede ser común. En la figura 14b se muestra el diagrama esquemático de un multiplexor dual de cuatro entradas (MUX) con un solo habilitador.

La NAND es la forma de compuerta preferida para muchos CI (por ejemplo, el 74LS00 y el 74LS10). Puesto que el diseño del multiplexor en la figura 13 o 14 es un circuito AND-OR de dos niveles, la sustitución directa de todas las compuertas AND y OR por compuertas NAND mantendrá la función lógica, como se explicó en el capítulo anterior. De este modo, la implementación real del multiplexor se efectúa con compuertas NAND.

Multiplexores como circuitos lógicos de propósito general

Es claro que la estructura de un multiplexor en las figuras 13 y 14 es la de un circuito lógico AND-OR de dos niveles, teniendo cada compuerta AND $n + 1$ entradas, donde n es el número de entradas de selección. Parece que el multiplexor constituiría una implementación canónica de suma de productos de una función de conmutación si todas las líneas de datos en conjunto

representan jiisto una variable de conmutación (o su complemento) y cada una de las entradas de selección, una variable de conmutación.

Vamos a trabajar hacia atrás desde una función especificada de m variables de conmutación para la cual tenemos escrita una expresión canónica de suma de productos. El tamaño del multiplexor necesario (numero de entradas de selección) no es evidente. Suponga que elegimos un multiplexor que tiene $m - 1$ entradas de selección, dejando únicamente otra variable más para acomodar todas las entradas de datos. Escribimos una función de salida de estas entradas de selección y las 2^{m-1} entradas de datos D_i . Ahora planeamos asignar $m - 1$ de estas variables a las entradas de selección; ¿pero cómo hacer la asignación?⁴ Realmente no hay restricciones, por lo que puede realizarse de manera arbitraria.

El siguiente paso es escribir la salida del multiplexor después de las entradas de selección con $m - 1$ de las variables de la función dada. Comparando las dos expresiones término por término, las entradas D_i pueden determinarse en términos de la variable restante.

EJEMPLO 1

Una función de conmutación que se va a implementar con un multiplexor es:

$$f(x, y, z) = \Sigma(1, 2, 4, 7) = x'y'z + x'yz' + xy'z' + xyz$$

Puesto que la función tiene tres variables, el multiplexor deseado tendrá $3 - 1 = 2$ entradas de selección; la mitad del MUX dual de cuatro entradas de la figura 14 realizará la tarea. La expresión para la salida del multiplexor es:

$$f = s_1's_0'D_0 + s_1's_0D_1 + s_1s_0'D_2 + s_1s_0D_3$$

No hay restricciones acerca de cómo asignar las entradas de selector a las variables de la función dada; dejemos arbitrariamente que $s_1 = x$ y $s_0 = y$. Entonces

$$f = x'y'D_0 + x'yD_1 + xy'D_2 + xyD_3$$

Comparando ésta con la expresión original para la función dada llegamos a

$$D_0 = D_3 = z$$

$$D_1 = D_2 = z'$$

Así, la función original se implementa con un multiplexor de cuatro entradas. □

Existen otras cinco maneras mediante las cuales las dos entradas de selección podrían haber sido asignadas a dos de las tres variables de conmutación. Ninguna condición necesita cumplirse para la elección, por lo que ésta es arbitraria. Sin embargo, el resultado específico obtenido para las entradas D_i depende de esa elección inicial.

Ejercicio 7. En el problema del ejemplo 1, elija $s_1 = z$ y $s_0 = x$. Determine las D_i .

Respuesta⁵

Ejercicio 8. Como practica, elija cada una de las restantes maneras posibles de asignar entradas de selección a las variables de conmutación, y determine después las D_i requeridas; especifique las compuertas externas necesarias. •

Para un conjunto de $m - 1$ variables, hay m maneras de asignar $m - 1$ cantidades a variables específicas!

⁵ $D_0 = D_3 = y$, $D_1 = D_2 = y'$

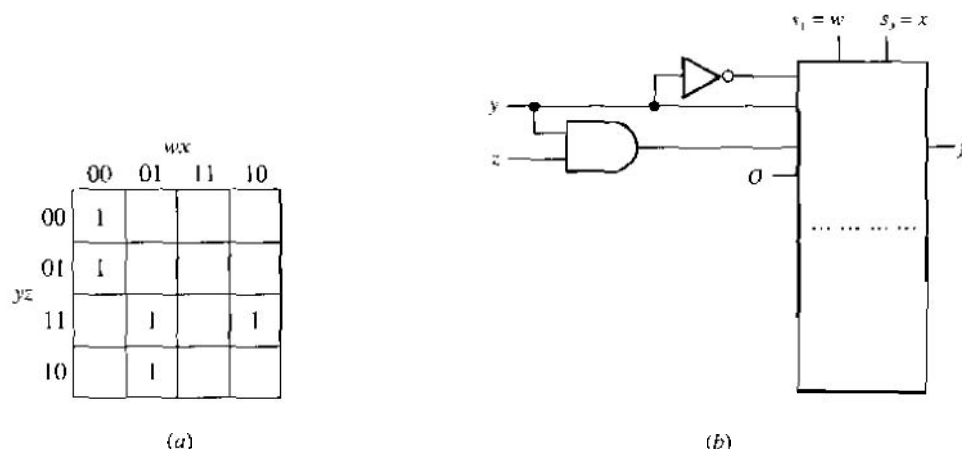


Figura 15. Implementación de multiplexor de $f = \Sigma(0, 1, 6, 7, 11)$.

Para implementar una función de conmutación de m variables, hemos visto que un multiplexor de $m - 1$ entradas de selección puede realizar la función.

Podría ser posible en algunos casos el empleo de un multiplexor incluso más pequeño. Debe esperarse que estos ahorros en la complejidad del MUX deben originarse a expensas de algún otro costo.

EJEMPLO 2

La función de cuatro variables cuyo mapa se muestra en la figura 15, se va a implementar mediante un multiplexor. Siempre es posible uno con $4 - 1 = 3$ variables de selección. Sin embargo, vamos a explorar la posibilidad de utilizar un multiplexor con sólo dos variables de selección para implementar esta función.

Arbitrariamente asignamos las dos entradas de selección s_1 y s_0 a w y x . La expresión para la salida del multiplexor es la misma que la que se indicó en el ejemplo 1, ya que ésta tiene las mismas dimensiones. Para $wx = s_1s_0 = 00$, esa expresión se reduce a D_0 . Pero para los valores $wx = 00$, la expresión que cubre los 1 en el mapa es $y'z' + y'z = y'$. Por consiguiente, $D_0 = y'$. De manera similar, en la columna 01 del mapa, la expresión se reduce a $D_1 = y$ y el mapa produce $yz + yz' = y$; por consiguiente, $D_1 = y$. De la misma manera, de la columna 11 encontramos $D_2 = 0$ y de la columna 10, $D_3 = yz$. (Confirme esto.) En la figura 15b se muestra este circuito que es bastante simple. Encontramos que para implementar cierta función específica de cuatro variables, puede emplearse un multiplexor de orden menor que 3, a costa de una compuerta AND adicional. (El inversor sería necesario incluso con un multiplexor de orden superior, por lo que no cuenta como costo agregado.) ■

Ejercicio 9. En el ejemplo anterior, suponga que s_1 y s_0 se identifican como y y z en vez de w y x . Determine expresiones para las entradas de datos en términos de w y x , y especifique el hardware externo que se necesitará además del multiplexor. Advierta la diferencia en complejidad para las dos elecciones de entradas de selección.

Respuesta^b

^b $D_0 = D_1 = w'x'$, $D_2 = w'x$, $D_3 = w \oplus x$; tres compuertas AND y una compuerta XOR, además de un MUX de cuatro entradas.

En la implementación de una función de conmutación arbitraria, distintas elecciones de las entradas de selección conducen a cantidades diferentes de hardware externo para un multiplexor más pequeño que lo normal. Desafortunadamente, a pesar de haberlas probado, no hay forma de determinar cuál elección será la **más** económica.

3 DECODIFICADORES Y CODIFICADORES

La sección anterior se inició explicando una aplicación: dadas 2^n señales de datos, el problema es elegir, bajo el control de n entradas de selección, secuencias de estas 2^n señales de datos para enviarlas serialmente por un enlace de comunicaciones. La operación inversa en el extremo receptor del enlace de comunicación consiste en recibir los datos serialmente en una sola línea y transmitirlos a una de 2^n líneas de salida. Esto se controla de nuevo por medio de un conjunto de entradas de control. Esta aplicación es la que necesita únicamente una línea de entrada; otras aplicaciones quizá requieran más de una. A continuación investigaremos un circuito generalizado de este tipo.

De modo concebible, podría haber un circuito combinatorio que acepta n entradas (no necesariamente 1, aunque un número pequeño) y que ocasiona que los datos se direccionen hacia una de muchas, digamos hasta 2^n salidas. Estos circuitos tienen el nombre genérico de *decodificador*. Al menos semánticamente, si algo se va a decodificar, previamente debe haberse *codificado*, que es la operación inversa de la decodificación. Al igual que un multiplexor, un circuito codificador debe aceptar datos de gran número de líneas de entrada y convertirlos en datos en un número más pequeño de líneas de salida (no necesariamente una sola). Esta sección explicará varias implementaciones de decodificadores y codificadores.

Demultiplexores

Refiérase al diagrama de la figura 12. El demultiplexor que se muestra ahí corresponde a un circuito de una sola entrada y de salida múltiple. Sin embargo, además de la entrada de datos debe haber otras entradas para controlar la transmisión de estos últimos hacia la línea de salida de datos apropiada en algún tiempo determinado. En la figura 16a se presenta un circuito demultiplexor de este tipo que tiene ocho líneas de salida. Es instructivo comparar este circuito demultiplexor con el circuito multiplexor de la figura 13. Para el mismo número de entradas de control (selección), existe el mismo número de compuertas AND. Pero en este caso cada salida de compuerta AND es una salida de circuito. En vez de que cada compuerta tenga su propia entrada de datos independiente, la línea de datos única constituye ahora una de las entradas de cada compuerta AND. Las otras entradas de las compuertas AND son líneas de control.

Cuando la palabra formada por las entradas de control $C_2C_1C_0$ es el equivalente binario del decimal k , entonces la entrada de datos x se dirige a la salida D_k . Viéndolo de otra forma, en un demultiplexor con n entradas de control, cada salida de compuerta AND corresponde a 1 minitérmino de n variables. Para una combinación determinada de entradas de control, sólo un minitérmino puede tomar el valor 1; la entrada de datos se dirige a la compuerta AND correspondiente a este minitérmino. Por ejemplo, la expresión lógica para la salida D_3 es $xC_2'C_1C_0$. En consecuencia, cuando $C_2C_1C_0 = 011$, entonces $D_3 = 1$ y todas las otras D_i son 0. La tabla de verdad completa para el demultiplexor de ocho salidas se muestra en la figura 16b.

Decodificador de n a 2^n líneas

Suponga que se elimina la línea de entrada de datos en el circuito demultiplexor de la figura 16. (Dibuje el circuito usted mismo.) Cada compuerta AND cuenta ahora con sólo n (en este caso tres) entradas, y hay 2^n (en este caso ocho) salidas. Puesto que no hay una línea de entrada de

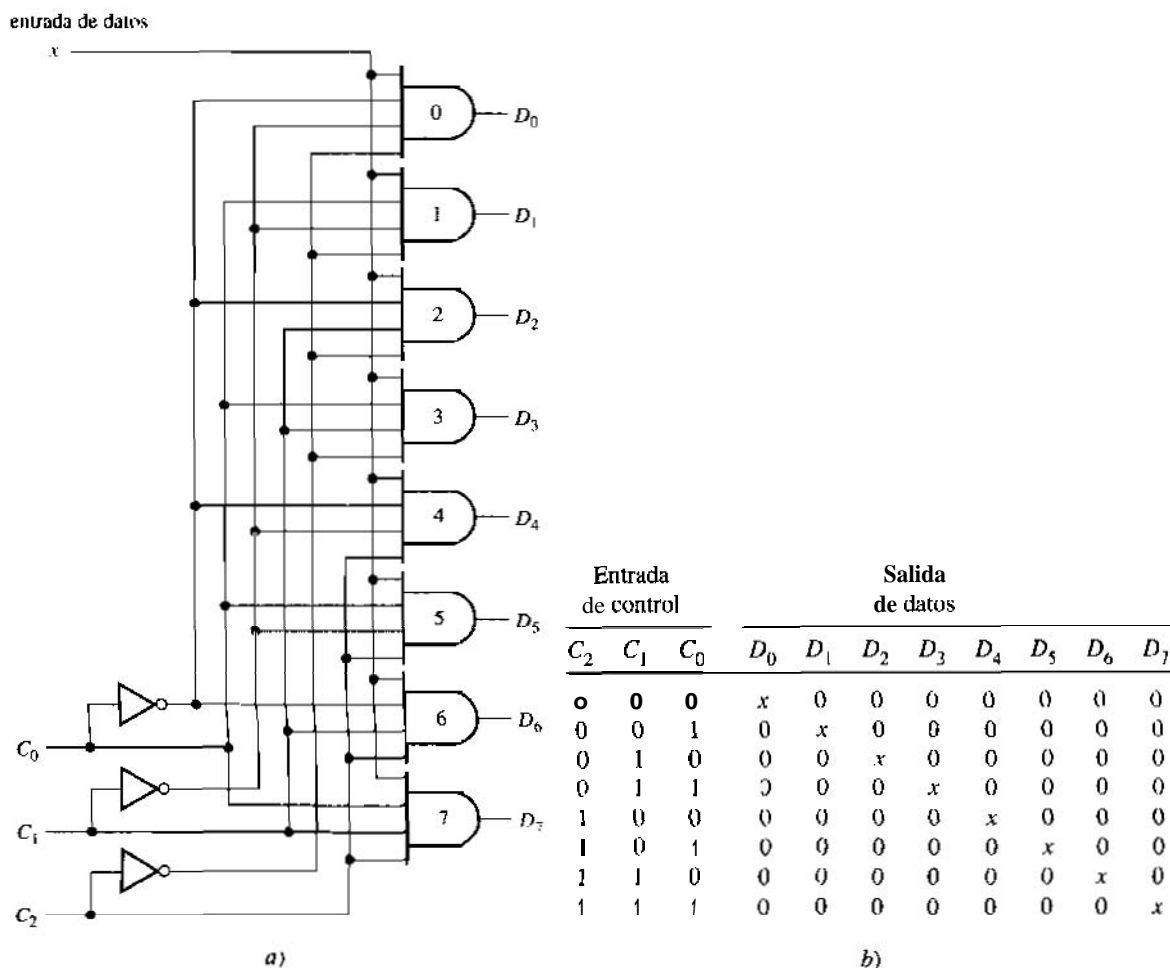


Figura 16. Un circuito demultiplexor a) y su tabla de verdad b)

datos que controlar, lo que usamos como entradas de control ya no sirven para esa función. En vez de eso, éstas son las entradas de datos que se van a decodificar. Este circuito es un ejemplo de lo que se llama un *decodificador de n a 2^n líneas*. Cada salida representa un minitérmino. La salida k es 1 siempre que la combinación de los valores de las variables de entrada corresponde al equivalente binario del decimal k .

Ahora suponga que no se elimina la línea de entrada de datos del demultiplexor de la figura 16 sino que se retiene y se observa como una entrada habilitadora. El decodificador opera en este caso solo cuando la x habilitadora es 1. Visto de manera inversa, un decodificador de n a 2^n líneas con una entrada habilitadora también puede utilizarse como un demultiplexor, donde el habilitador se convierte en la entrada de datos en serie y las entradas de datos del decodificador vienen a ser las entradas de control del demultiplexor.⁷

Los decodificadores del tipo que acaba de describirse se disponen como circuitos integrados (MSI); $n = 3$ y $n = 4$ son bastante comunes.

⁷ En la práctica, la implementación física del decodificador con habilitador se efectúa con compuertas NAND. En ese caso, lo que se obtiene son los complementos de las salidas en el circuito que se analiza, y la entrada habilitadora se invierte antes si se aplica a las compuertas NAND. Estos son detalles prácticos que no cambian los principios descritos aquí.

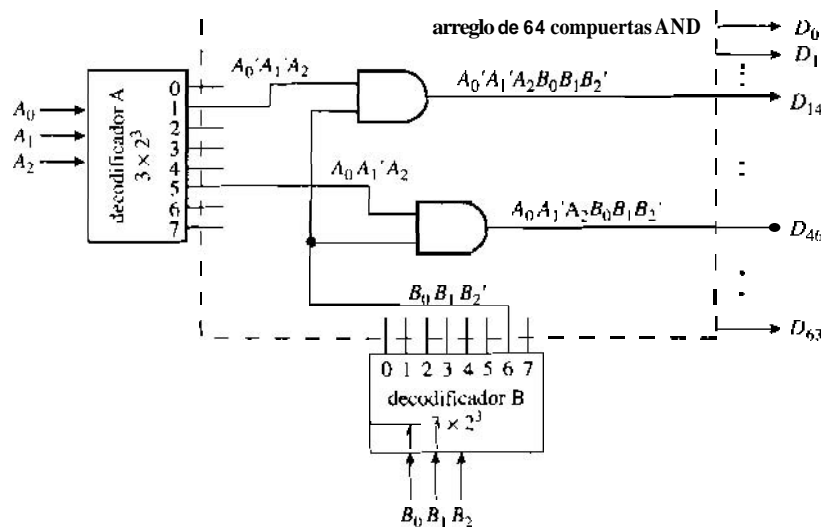


Figura 17. Diseño de un decodificador de 6 a 2^6 líneas a partir de dos decodificadores de 3 a 2^3 líneas con una matriz de interconexión de 64 compuertas AND.

No hay **razón teórica** por la que n no pueda aumentarse a valores superiores. Sin embargo, puesto que siempre habrá limitaciones prácticas en el factor de carga de entrada (el número de entradas que una compuerta física puede soportar), los decodificadores de orden superior **se diseñan muchas veces** utilizando decodificadores de orden inferior interconectados con una red de otras compuertas.

En la figura 17 se presenta una ilustración del diseño de un decodificador de 6 a 2^6 líneas construido a partir de dos decodificadores de 3 a 2^3 líneas. Cada uno de estos últimos tiene ocho salidas. Cada una de las salidas provenientes del decodificador A debe ser multiplicada con cada una de las salidas del decodificador B para producir una de las 64 salidas del decodificador completo mediante compuertas AND. De ese modo, además de las 8 compuertas AND de tres entradas en cada decodificador de 3 a 2^3 líneas, hay 63 compuertas AND de dos entradas en la red de interconexión. Sólo dos de éstas se indican explícitamente en la figura 17.

Ejercicio 10. Diseñe un decodificador de 6 a 2^6 líneas utilizando la estructura de la figura 16. Especifique el número de compuertas AND y el número total de líneas de entrada de todas las compuertas. Compare con el diseño de la figura 17. ♦

Decodificador de árbol

Cuando se diseñan decodificadores de orden superior en una jerarquía de varias etapas de orden inferior, **se produce una dificultad práctica** con el factor de carga de salida (número de compuertas alimentadas por una terminal). (Por una jerarquía de etapas entendemos, por ejemplo, dos etapas de 3×8 para formar un codificador de 6×64 , como en la figura 17; después dos etapas 6×64 para formar un decodificador de 12×2^{12} ; etc.) Ya en la figura 17, cada compuerta en los decodificadores componentes actúa sobre otras ocho compuertas. En el siguiente nivel de la jerarquía, cada una de las salidas de las compuertas desde el nivel siguiente hasta el último tendrá que actuar sobre otras 64 compuertas.

Este problema se supera, aunque sólo parcialmente, mediante el diseño del decodificador tal como se ilustra en la figura 18. Esta configuración se conoce como **decodificador de árbol**. La primera etapa es un decodificador de 2 a 4 líneas. Se introduce una nueva variable en cada etapa

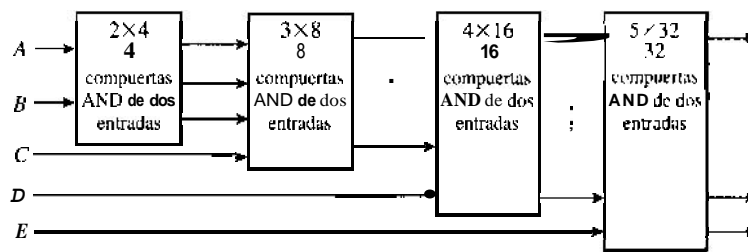


Figura 18. Diseño de un decodificador de árbol

sucesiva; ésta o su inversa se vuelve una entrada para cada una de las compuertas AND de dos entradas en esta etapa. La segunda salida a cada compuerta AND proviene de la etapa precedente. Por ejemplo, una de las salidas de la segunda etapa será $AB'C$. Esto dará origen a dos salidas de la siguiente etapa, $AB'CD$ y $AB'CD'$. Este diseño evita el problema del factor de carga de salida en las primeras etapas, pero no en las últimas. A pesar de eso, el problema existe sólo para las variables que se introdujeron en aquellas etapas. Todos los remedios que se requieran tendrán que utilizarse para un número relativamente pequeño de variables, en oposición al número considerable que requiere el diseño de la figura 17.

Decodificadores como circuitos lógicos de propósito general: conversión de código

Puesto que cada salida de un decodificador de n a 2^n líneas es un producto canónico de literales, simplemente la aplicación de una compuerta OR a todas las salidas produce una suma canónica de productos. Y ya que cualquier función de conmutación se puede expresar como una suma canónica de productos, se concluye que toda función de conmutación puede implementarse mediante un decodificador de n a 2^n líneas seguido por una compuerta OR. (Si 2^n excede la limitación del factor de carga de entrada de la compuerta OR, serán necesarios niveles adicionales de compuertas OR.) En realidad, si se va a implementar más de una función de las mismas variables, es posible utilizar el mismo decodificador, teniendo cada función su propio conjunto de compuertas OR.

Una clase fundamental de circuito lógico se conoce como *convertidor de código*. Se trata de un circuito que acepta como entrada los dígitos de una palabra que expresa alguna información en un código particular y que produce como salidas los dígitos de una palabra en un código diferente. (Véase el capítulo 1 para una introducción a los códigos.) Ilustraremos el uso de un decodificador como un convertidor de código diseñando un circuito para convertir del código de exceso 3 al código de siete segmentos. (Estos códigos se presentan en la figura 4 y en el ejercicio 12 del capítulo 1; se repiten aquí en la figura 19.)

Suponga que se dispone de un decodificador de 4 a 16 líneas. Puesto que sólo hay 10 palabras válidas de código de exceso 3, únicamente 10 de las 16 salidas de las compuertas AND adquieren alguna vez el valor de 1. De modo que sólo se usarán aquellas 10 salidas de un decodificador de 4 a 16 líneas. Éstas se indican en la figura 19 mediante sus equivalentes decimales.

La figura 19 es la tabla de verdad para cada una de las siete funciones de salida (las S_i) en términos de las cuatro variables de entrada. El circuito externo al decodificador consistirá de siete compuertas OR, una para cada segmento. Sólo necesita tomarse una decisión: ¿cuáles salidas del decodificador deben convertirse en las entradas a cada compuerta OR? Esto se responde para cada segmento listando los números de minterminos correspondientes a cada palabra de código para la cual esa salida de segmento tiene el valor 1. Las listas de minterminos para las salidas correspondientes a algunos de los segmentos son las siguientes:

Dígito decimal	Entradas: Exceso 3				Salidas: Siete segmentos						
	w	x	y	z	S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	S ₇
0	0	0	1	1	1	1	1	1	1	1	0
1	0	1	0	0	0	0	0	1	1	0	0
2	0	1	0	1	1	0	1	1	0	1	1
3	0	1	1	0	0	0	1	1	1	1	1
4	0	1	1	1	0	1	0	1	1	0	1
5	1	0	0	0	0	1	1	0	1	1	1
6	1	0	0	1	1	1	0	0	1	1	1
7	1	0	1	0	0	0	1	1	1	0	0
8	1	0	1	1	1	1	1	1	1	1	1
9	1	1	0	0	0	1	1	1	1	0	1

Figura 19. Conversión de código de exceso 3 a siete segmentos.

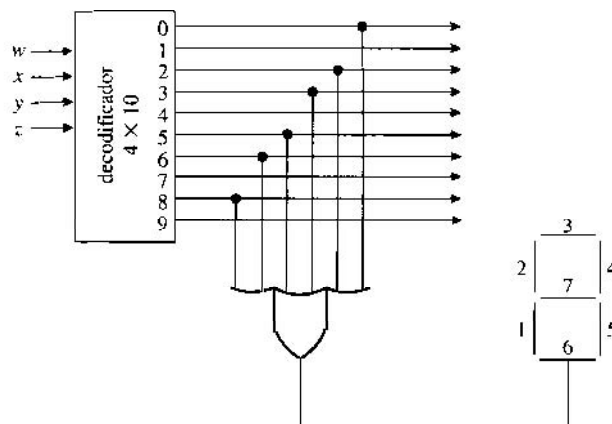


Figura 20. Convertidor de código de exceso 3 a siete segmentos.

$$S_1 = \Sigma(3, 5, 6, 8, 10, 11, 12)$$

$$S_4 = \Sigma(3, 4, 5, 6, 7, 10, 11, 12) \quad (8)$$

$$S_5 = \Sigma(3, 4, 6, 7, 8, 9, 10, 11, 12)$$

$$S_6 = \Sigma(3, 5, 6, 9, 11)$$

S610 una de las compuertas OR (la correspondiente a S_6) se muestra en la figura 20; debe haber otras seis. En ese caso, cuando una palabra de código de exceso 3 correspondiente a un dígito decimal aparece a la entrada, se iluminarán los segmentos apropiados, exhibiendo el dígito.

Ejercicio 11. Escriba la lista de minitérminos para los tres segmentos cuyas listas de minitérminos no se indicaron en 8). Confirme las entradas a la compuerta OR en la figura 20. ♦

4 MEMORIA SOLO DE LECTURA (ROM)

Un circuito para implementar una o más funciones de conmutación de varias variables se describió en la sección precedente y se ilustró en la figura 20. Los componentes del circuito son:

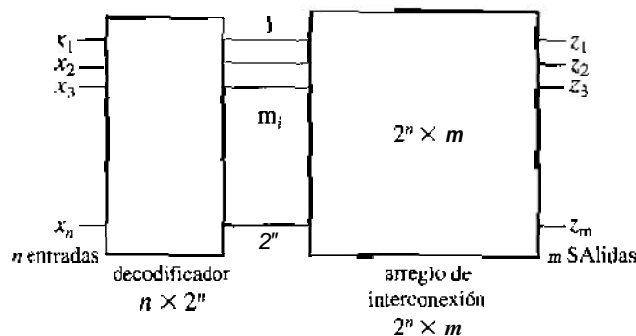


Figura 21. Estructura básica de una ROM.

- Un decodificador de $n \times 2^n$, con n líneas de entrada y 2^n líneas de salida.
- Una o más compuertas OR, cuyas salidas son las salidas del circuito.
- Una red de interconexión entre salidas del decodificador y entradas de compuerta OR.

El decodificador es un circuito MSI, compuesto por 2^n compuertas AND de n entradas, que produce todos los minterminos de n variables. Éste proporciona cierta economía de realización, debido a que el mismo decodificador puede usarse para cualquier aplicación que implique el mismo número de variables. Para una aplicación particular sólo cambia el número de compuertas OR y las salidas específicas del decodificador que se convierten en entradas para dichas compuertas OR. Será bienvenido todo lo que se haga para producir un circuito de propósito general.

El método de propósito general más común consiste en incluir el número máximo de compuerta OR, tomando provisiones para interconectar el total de las 2^n salidas del decodificador con las entradas de cada una de las compuertas OR. En esa situación, para cualquier aplicación dada, podrían presentarse dos cosas:

- El número de compuertas OR utilizadas podría ser menor que el número máximo, quedando las restantes sin uso.
- No todas las salidas del decodificador se tendrían que conectar a todas las entradas de las compuertas OR.

Este esquema sería terriblemente derrochador y no parece buena idea.

En vez de eso, suponga que se selecciona un número más pequeño, m , para el número de compuertas OR que se van a incluir, y que se establece una red de interconexión para interconectar las 2^n salidas del decodificador a las m entradas de las compuertas OR. Esta estructura se ilustra en la figura 21; corresponde a un circuito combinatorio LSI con n entradas y m salidas que, por razones que serán claras más adelante, se conoce como memoria de sólo lectura (ROM). Una ROM consta de dos partes:

- Un decodificador de $n \times 2^n$.
- Un arreglo de $2^n \times m$ dispositivos de conmutación que forman interconexiones entre las 2^n líneas del decodificador y las m líneas de salida.

Las 2^n líneas de salida del decodificador se denominan líneas de palabra. Cada una de las 2^n combinaciones que constituyen las entradas al arreglo de interconexión corresponden al mintermino y especifican una dirección. La memoria consta de aquellas conexiones que se efectúan realmente en la matriz de conexión entre las líneas de palabra y las líneas de salida. Una vez hechas, las conexiones en el arreglo de memoria son permanentes.⁸

⁸ En ciertos discos, es posible que las conexiones sean horribles; esto se describirá más adelante.

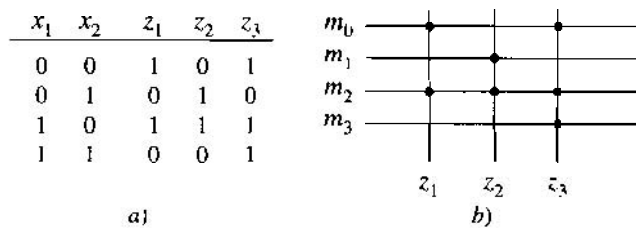


Figura 22. Una tabla de verdad ROM y su programa.

De modo que esta memoria no es una cuyos contenidos puedan cambiarse con facilidad de un tiempo a otro: sólo "escribimos" en esta memoria una vez. Sin embargo, es posible "leer" la información ya almacenada (las conexiones ya hechas) tan a menudo como se desee, aplicando palabras de entrada y observando las palabras de salida. Ésta es la razón por la que el circuito recibe el nombre de *memoria de sólo lectura*.⁹

Antes de continuar, piense en dos maneras posibles de fabricar una ROM de modo que un conjunto de conexiones pueda efectuarse y el otro conjunto quede **desconectado**. Continúe después de haber pensado acerca de ello.

La "escritura" de un tiempo en la memoria puede ejecutarse del modo siguiente:

- Una ROM puede fabricarse casi por completo, dejando pendiente solamente la realización de todas las conexiones. Se dice que una ROM de este tipo estará en *blanco*. La formación de las conexiones para una aplicación particular se denomina *programación* de la ROM. En el proceso de programación de la ROM, se produce una *máscara* para cubrir aquellas conexiones que no se van a efectuar. Por esta razón, la forma en blanco de la ROM se conoce como *máscara programable*.¹⁰
- Una ROM puede fabricarse completamente de modo tal que todas las conexiones potenciales se hayan realizado. Una ROM de este tipo estará en blanco. La programación de la ROM para una aplicación específica consiste en abrir aquellas conexiones que son indeseables. En este caso, la ROM en blanco se dice que será *programable en campo* (denominada PROM). Las conexiones se efectúan colocando un *fusible* o *enlace* en cada punto de conexión. En cualquier aplicación específica, se abren o "apagan" las conexiones indeseables haciendo pasar pulsos de corriente a través de ellas. Una medida del costo de la PROM es el número de enlaces de fusible. $2^n \times m$.¹¹

Una vez que se ha programado la ROM, una palabra de entrada x_1, x_2, \dots, x_n activa una línea de palabra específica correspondiente al minitérmino formado por los valores específicos de las x_i . Las conexiones en la matriz de salida producen la palabra de salida deseada.

EJEMPLO 3

La figura 22a presenta una tabla de verdad para la matriz de interconexión de una ROM de $2^2 \times 3$. La tabla de verdad implica el programa ROM representado mediante los puntos en las in-

⁹ Aunque aparece en su nombre la palabra "memoria", una ROM no cuenta con una memoria en el sentido usual. Como se describirá en los capítulos 5 y 6, la memoria se caracteriza por circuitos secuenciales, mas no combinatorios.

¹⁰ La máscara, que requiere atención minuciosa, implica una costosa producción. Por consiguiente, las ROM de programación por máscara se usan sólo cuando un número muy grande de lotes de producción justifica el costo.

¹¹ Algunas PROM se fabrican de manera que sea posible restituir las a su condición en blanco después de que se han programado para una aplicación específica: éstas son PROM borrables, o EPROM. Tienen una clara ventaja sobre el tipo no borrable, pero su costo es correspondientemente superior.

tersecciones de las líneas de palabras de entrada y salida en la figura 22b. Cada palabra de entrada define una palabra de salida, como lo requiere la tabla de verdad. Si la palabra de entrada es 01 (correspondiendo al minitérmino m_1), por ejemplo, sólo se activará la línea de salida z_2 , pues es la única conexión con m_1 en la matriz de conexión. Por consiguiente, la palabra de salida será 010, como se confirma también en la tabla de verdad. (Verifique a partir de la tabla de verdad que el resto del programa es correcto.)

Ejercicio 12. Se va a programar una ROM para implementar la conversión del código de exceso 3 al de siete segmentos cuya tabla se indicó en la figura 19. Las ROM se presentan en laminares estándar, y $m = 7$ no es uno de ellos. El siguiente tamaño estándar más grande es $m = 8$. En consecuencia, la tabla de verdad tendrá seis renglones más y una columna más que lo que se indica en la figura 19. (Especifique cuáles serán las entradas en la tabla de verdad para estos renglones y columnas adicionales.) Dibuje el número apropiado de líneas de cruce para las palabras de entrada y salida. Recurriendo a la tabla de verdad, programe la ROM poniendo puntos en las intersecciones apropiadas de estas dos palabras.

En el ejercicio 12 el número de entradas en la tabla de verdad (que corresponde al número de enlaces entre las palabras de entrada y de salida) es $2^n \times M = 16 \times 8 = 128$. De éstas, la mitad representa valores no relevantes. Hay casos bastante peores que éste; en ocasiones se usa apenas 1% de los enlaces, lo que origina un "desperdicio" considerable en este tipo de implementaciones ROM. Una implementación que evite este desperdicio sería bienvenida, lo cual es el tema de la sección siguiente.

5 OTROS DISPOSITIVOS LÓGICOS PROGRAMABLES LSI

Una forma de considerar la ROM que se explicó en la sección anterior corresponde a un dispositivo con una estructura específica (un conjunto de compuertas AND y un conjunto de compuertas OR) que el diseñador puede utilizar para obtener las salidas deseadas efectuando unas cuantas modificaciones. Podríamos decir que la ROM ha sido "programada" para producir sus salidas específicas. Existen otras estructuras que tienen esta propiedad, a saber, programabilidad. Un nombre genérico para ellas es *dispositivo lógico programable* (o programado) (PLD).

La ROM implementa funciones lógicas como suma de minitérminos. Para n variables de entrada hay 2^n minitérminos y, en consecuencia, 2^n compuertas AND, cada una con n entradas. Como acaba de explicarse, con frecuencia en muchas de las funciones lógicas, no se usan muchas de las compuertas AND ni los enlaces que las conectan a las compuertas OR de salida. A continuación explicaremos dos implementaciones en las cuales se evita parte de este desperdicio.

Arreglo lógico programado (PLA)

La implementación canónica de suma de productos de una función lógica constituye un desperdicio en dos formas: en el número de compuertas AND utilizadas (tantas como los minitérminos que hay, 2^n) y en el número de entradas a cada compuerta AND (n). Supóngase que consideramos una implementación reducida (posiblemente mínima) de suma de productos. Dada una función lógica de n variables, el número más grande de términos en una expresión mínima de suma de productos que representa esta función es 2^{n-1} , justo la mitad del número de minitérminos. (Véase el problema 36 en el capítulo 3.) Eso equivale a ahorro de 50% en compuertas AND para el peor caso de una salida. Puesto que habrá un conjunto reducido de entradas a las compuertas AND, este ahorro en compuertas se paga por la necesidad de programar no sólo las salidas de las compuertas AND sino sus entradas también. La estructura del circuito que se produce recibe el nombre de *arreglo lógico programable* (o programado) (PLA). Éste se ilustra en la figura 23 para el caso de $n = 3$ variables de entrada, $m = 4$ funciones de salida, y cuatro compuertas AND.

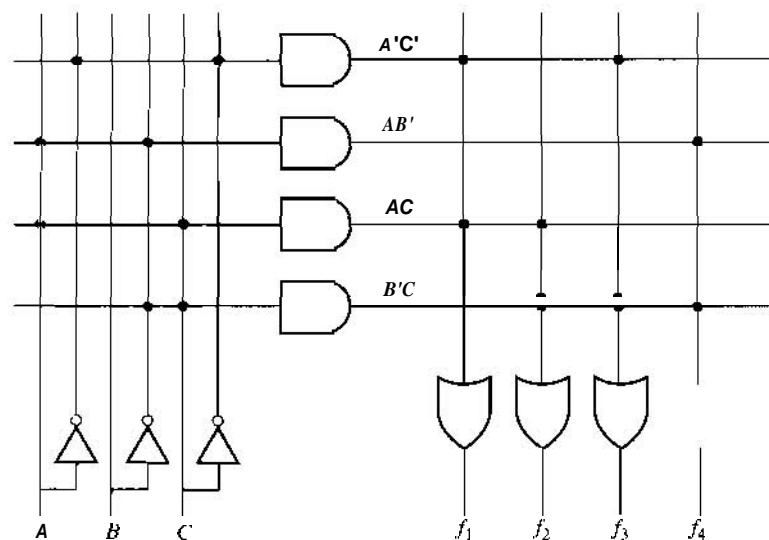


Figura 23. Estructura de un PLA.

El diagrama en la figura 23 no corresponde al diagrama clásico de un circuito. Se muestra una sola línea para representar todas las entradas a cada compuerta AND y OR. El número de líneas de entrada a cada compuerta AND debe ser $2n$, el doble del número de entradas, para ajustar la posibilidad de conectar cada variable o su complemento a cada compuerta AND. El número de líneas de entrada a cada compuerta OR debe ser igual al número de compuertas AND, digamos p . (Por simplicidad y sin peligro de confusión, los símbolos de compuerta pueden omitirse.) Las conexiones programadas entre las entradas y las compuertas AND, y entre las salidas de las compuertas AND y de las compuertas OR para un conjunto específico de funciones de salida, se muestran por medio de los puntos en las intersecciones.

Los mapas de las cuatro funciones de salida y de las expresiones mínimas de suma de productos se presentan en la figura 24. En este ejemplo, un total de sólo cuatro términos producto abarca todas las funciones, de modo que se necesitan únicamente cuatro compuertas AND en la realización. Dos conjuntos de líneas deben programarse: las líneas de entrada y las líneas de salida. Para efectuar esto, construimos una tabla de programación como sigue:

- Los implicantes (términos producto) se listan como encabezados de renglones.
- En un conjunto de columnas, los encabezados son las variables de entrada; esta parte de la tabla debe proporcionar la información que indica cuáles variables o (sus complementos) son factores en cada implicante.
- En un segundo conjunto de columnas, los encabezados son las funciones de salida; esta parte de la tabla debe dar la información que indica la compuerta de salida a la cual se dirige cada implicante (salida de compuerta AND).

En el primer conjunto de columnas, si ésta presenta una variable (no complementada) en un renglón particular, la entrada correspondiente es 1; si su complemento está presente, la entrada es 0. Si ninguno aparece, la entrada puede dejarse en blanco, aunque es preferible mostrar algún símbolo sustituto; a menudo se emplea una raya.

En el segundo conjunto de columnas, que corresponde a las funciones de salida, si una función particular cubre un implicante particular, entonces la entrada correspondiente es 1; en otro caso podría dejarse en blanco, pero suele anotarse un punto. Como ejemplo, considere el renglón 4. Puesto que el implicante es $y'z$, la entrada en la columna z es 1, la correspondiente en la columna y es 0, y la relativa a x es una raya. En las columnas de salidas, únicamente f_1 no cubre

		x	
		0	1
yz	00	1	
	01		1
	11		1
	10	1	

f_1

		x	
		0	1
yz	00		
	01	1	1
	11		1
	10		

f_2

		x	
		0	1
yz	00	1	
	01	1	1
	11		
	10	1	

f_3

		x	
		0	1
yz	00	1	
	01	1	1
	11		
	10		

f_4

Término producto	Entradas			Salidas				
	y	z		f_1	f_2	f_3	f_4	
1: $x'z'$	0	—	0	1	•	1	•	$f = x'z' + xz$
2: xy'	1	0	—	•	•	•	1	$f_2 = xz + y'z$
3: xz	1	—	1	1	1	•	•	$f_3 = x'z' + y'z$
4: $y'z$	—	0	1	•	1	1	1	$f_4 = xy' + y'z$

Figura 24. Programación del PLA.

al implicante $y'z$; por tanto, la entrada será 1 en toda columna en el renglón 4 excepto la f_1 , donde la entrada es 0. Confirme lo que ocurre en el resto de los renglones.

Una vez que se efectúa la programación, la fabricación de los enlaces (puntos de conexión) en un PLA se lleva a cabo de manera similar a la de la ROM. El PLA es programable por máscara o programable en campo (FPLA). En el caso del FPLA, con p = el número de compuertas AND, existirán $2np$ enlaces en las entradas y mp enlaces en las salidas.

Para el ejemplo de la figura 23, el número de enlaces es igual a $4(6 + 4) = 40$. Sólo 16 de éstos se conservan, lo que significa que 24 enlaces tienen que abrirse durante la programación de campo. Los PLA típicos tienen más entradas, salidas y compuertas AND que los que se indican en el ejemplo de la figura 23. (El CI tipo 82S100, por ejemplo, incluye $n = 16$, $m = 8$ y $p = 48$.)

Cuando un conjunto de funciones de conmutación se presenta para implementarse con un PLA, una meta de diseño sería la reducción de p (el número de compuertas AND). La economía que se alcanza no proviene de una reducción en los costos de producción de las compuertas. (El costo de producción de un CI es prácticamente el mismo para uno con M compuertas que para uno con 50.) En vez de eso, la eliminación de una compuerta AND hace a un lado $2n = m$ enlaces; la principal fuente de ahorros es la eliminación del sustancial número de enlaces debido a la eliminación de cada compuerta AND. Por otro lado, la reducción del número de compuertas AND hasta un mínimo no significa que cada función se minimizará o que todos los implicantes serán primos. Los implicantes deben elegirse de manera que la mayor cantidad posible de ellos sea común a muchas de las funciones de salida.

Lógica de arreglo programado (PAL)

Una ROM tiene gran número de enlaces de fusible ($m \times 2^n$) debido al gran número (2^n) de compuertas AND. La programación de los enlaces se efectúa sólo sobre las salidas de las compuertas AND. En un PLA, el número de enlaces se reduce considerablemente al disminuir el número de compuertas AND. Lo último se lleva a cabo cambiando la expresión que representa a la función de conmutación de una forma canónica de suma de productos a una suma de productos en

Término producto		Entradas												Salidas					
Numero	Función	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6
1														•	•	•	•	•	1
2														•	•	•	•	•	1
3														•	•	•	•	•	1
4														•	•	•	•	•	1
5														•	•	•	•	1	•
6														•	•	•	•	1	•
7	$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}$	1	0	-	-	1	-	1	-	-	-	0	1	•	•	•	1	•	•
8														•	•	•	1	•	•
9														•	•	1	•	•	•
10														•	•	1	•	•	•
11														•	1	•	•	•	•
12														•	1	•	•	•	•
13														1	•	•	•	•	•
14														1	•	•	•	•	•
15														1	•	•	•	•	•
16														1	•	•	•	•	•

Figura 25. Tabla de programación para un ejemplo de PAL.

menor número de términos. El costo está en la necesidad de programar no sólo las salidas de las compuertas AND, sino también las entradas a estar mismas compuertas. ¿Qué otra posibilidad de programación existe más allá de los dos casos relativos a a) programar las salidas de las compuertas AND y b) programar tanto las entradas como las salidas? Estamos seguros de que responderá: "programar exclusivamente las entradas". Ésta es una posibilidad. ¿pero vale la pena!

En el caso de la ROM, no hay necesidad de programar las entradas debido a que, para cualquier función de n variables, existirá el mismo (gran) número de compuertas AND. De la misma manera, si pudiera fijarse el número de compuertas OR a la salida, entonces sería posible evitar la programación de las salidas de las Compuertas AND.

En muchos circuitos con salidas múltiples, aun cuando las salidas son funciones de un número mayor de variables de entrada, el número de términos producto en cada salida es pequeño. Por ello es pequeño el número de compuertas AND que llegan a cada compuerta OR. En tales casos, fijar permanentemente el número de compuertas OR y dejar únicamente la programación de las entradas de las compuertas AND para el diseño individual podría tener sentido en cuanto a la economía. El circuito resultante recibe el nombre de *lógica del arreglo programado* (PAL).¹² Existen PAL estándar para un número de valores bajos de p . Por ejemplo, el PAL16L8 tiene un máximo de 16 entradas y 8 salidas.

Una tabla de programación para un PAL es similar a la correspondiente a un PLA. Un caso con seis salidas se ilustra en la figura 25. Una ROM con 12 variables de entrada requeriría $2^{12} = 4096$ compuertas AND.

Sin embargo, vamos a suponer que en algunos casos posibles, la expresión canónica de suma de productos puede reducirse hasta 16 implicantes, indicándose sólo uno de ellos en la figura 25. Las entradas en la tabla tendrían el mismo significado que aquellas para el PLA. Sin embargo, para el PAL, las columnas de salida las fijaría el fabricante según el número de compuertas AND ya conectadas a cada compuerta OR.

Eri el caso presente, dos de las compuertas OR de salida son activadas cada una por cuatro compuertas AND; a cada una de las cuatro compuertas restantes las activan dos compuertas AND. En cualquier problema de diseño determinado, el primer paso es obtener una expresión

¹² PAL es una marca comercial registrada de Advanced Micro Devices.

apropiada de suma de productos, igual que en el caso de implementación PLA. Las conexiones de entrada se indican en la tabla como en el caso del PAL: una entrada es 1 si una variable aparece no complementada en un implicante, un 0 si aparece complementada, y una raya si no aparece en lo absoluto. Esto se ilustra para un renglón en la figura 25. El número de enlaces de fusible en este ejemplo es $2 \times 12 \times 16 = 384$. Esto es 20% menor que el número de enlaces de un PLA que tiene las mismas dimensiones. Por lo común, sin embargo, los PLA tienen más compuertas AND y por ello, para un PAL, el número de enlaces comúnmente sería muchas veces superior que el número correspondiente a un PLA comparable.

Ejercicio 13. Suponga que dos de los renglones de entradas en la figura 25 son del siguiente modo:

0 1 0 - 0 - - 1 - - - -
1 1 1 - - 0 - - 1 1 - -

¿Cuáles son los términos producto correspondientes?

En el capítulo 8 se dedicará un poco más de atención a los PLD. Igualmente se considerará ahí el uso de lenguajes de descripción de hardware en los diseños utilizando dispositivos lógicos programables.

RESUMEN Y REPASO DEL CAPÍTULO

En el capítulo 3, los diseños se efectuaron con compuertas primitivas en circuitos SSI. Este capítulo avanzó el proceso de diseño hacia circuitos más complejos implementados en unidades MSI. Los temas incluidos fueron

- Sumador binario.
- Sumador completo.
- Sumador de acarreo propagado.
- Sumador de acarreo anticipado.
- Restador binario.
- Sumador y restador en complemento a dos.
- Sumador y restador en complemento a uno.
- Multiplexor.
- Entrada de datos.
- Entrada de selección.
- Implementación de circuito lógico de propósito general con multiplexores.
- Demultiplexores.
- Líneas de entrada de datos.
- Líneas de entrada de control.
- Decodificador.
- Decodificador de $n \times 2^n$ líneas.
- Decodificador de árbol.
- Implementación de circuitos lógicos de propósito general con decodificadores.
- Conversión de código.
- Memoria de sólo lectura (ROM).
- Decodificador de $n \times 2^n$.
- Arreglo de interconexión de $2^n \times m$.
- Programación de una ROM.
- ROM de máscara programable.
- ROM de campo programable.

- Dispositivo lógico programable (PLD).
- Arreglo lógico programado (PLA).
- Lógica del arreglo programado (PAL).

PROBLEMAS

- Analice cada uno de los circuitos de sumador completo que se muestran en la figura P1 y escriba expresiones para la salida de cada compuerta intermedia.
 - Obtenga expresiones lógicas para las salidas del circuito suma y acarreo.
 - Confirme que estas expresiones son equivalentes a las funciones de suma y acarreo en las ecuaciones (1) del texto.
- Diseñe un sumador de acarreo anticipado de 4 bits. En la ecuación (7) del texto para la función de acarreo, sea $i = 0$ y deje que j varíe de 0 a 4. Escriba las expresiones resultantes para C_1 , C_2 , C_3 y C_4 .
 - Construya el diagrama lógico para el acarreo anticipado de 4 bits cuyo diagrama esquemático se presenta en la figura 8.
- Multiplique un número binario de 4 bits $Y = y_3y_2y_1y_0$ por medio de un número binario de 3 bits $X = x_2x_1x_0$. Utilice dos sumadores de 4 bits y las compuertas que sean necesarias para implementar esta operación, y dibuje el diagrama correspondiente.
- Demuestre formalmente que si la variable propagada P_i para un suinador de acarreo anticipado se define como $A_i + B_i$ en lugar de $A_i \oplus B_i$, la salida de la suma y el acarreo del sumador se seguirán calculando de manera correcta. (De también una demostración informal.) ¿Cuál definición es mejor con fines de implementación?
- Diseñe un circuito para la detección del desbordamiento en el sumador/restador de complemento a uno que se muestra en la figura 11.
- Muestre las conexiones en un diagrama esquemático de un multiplexor dual de cuatro entradas para realizar las funciones de suma y acarreo de un sumador completo.
 - Repita utilizando un decodificador de 3 a 2^3 líneas.
- Realice cada una de las siguientes funciones utilizando un multiplexor 8×1
 - $f = \Sigma(0, 1, 10, 11, 12, 13, 14, 15)$
 - $f = \Sigma(0, 3, 4, 7, 10)$
 - $f = \Sigma(0, 3, 4, 6, 7, 8, 12)$
 - $f = \Sigma(1, 2, 5, 8, 11, 12, 14)$
- Realice cada una de las funciones del problema 7 utilizando la mitad de un multiplexor dual de 4×1 y el número mínimo de compuertas externas.
- Repita el problema 7 utilizando un decodificador de 3 a 2^3 líneas.
- Utilice un multiplexor dual de cuatro entradas para implementar cada uno de las siguientes pares de funciones con el menor número de compuertas externas.
 - $f_1 = \Sigma(0, 4, 5, 7, 9, 11)$, $f_2 = \Sigma(2, 3, 5, 6, 10, 13)$
 - $f_1 = \Sigma(0, 4, 7, 10, 12, 14, 15)$, $f_2 = \Sigma(2, 7, 8, 9, 12, 13, 14, 15)$
- Muestre cómo conectar un sumador MSI de cuatro bits que sirva como un convertidor de código BCD de exceso 5.
 - Repita utilizando un decodificador de 4 a 10 líneas (BCD a decimal) y cuatro compuertas AND.
- Diseñe un decodificador BCD a decimal utilizando dos decodificadores de 2 a 3 líneas y un mínimo de compuertas AND interconectadas.

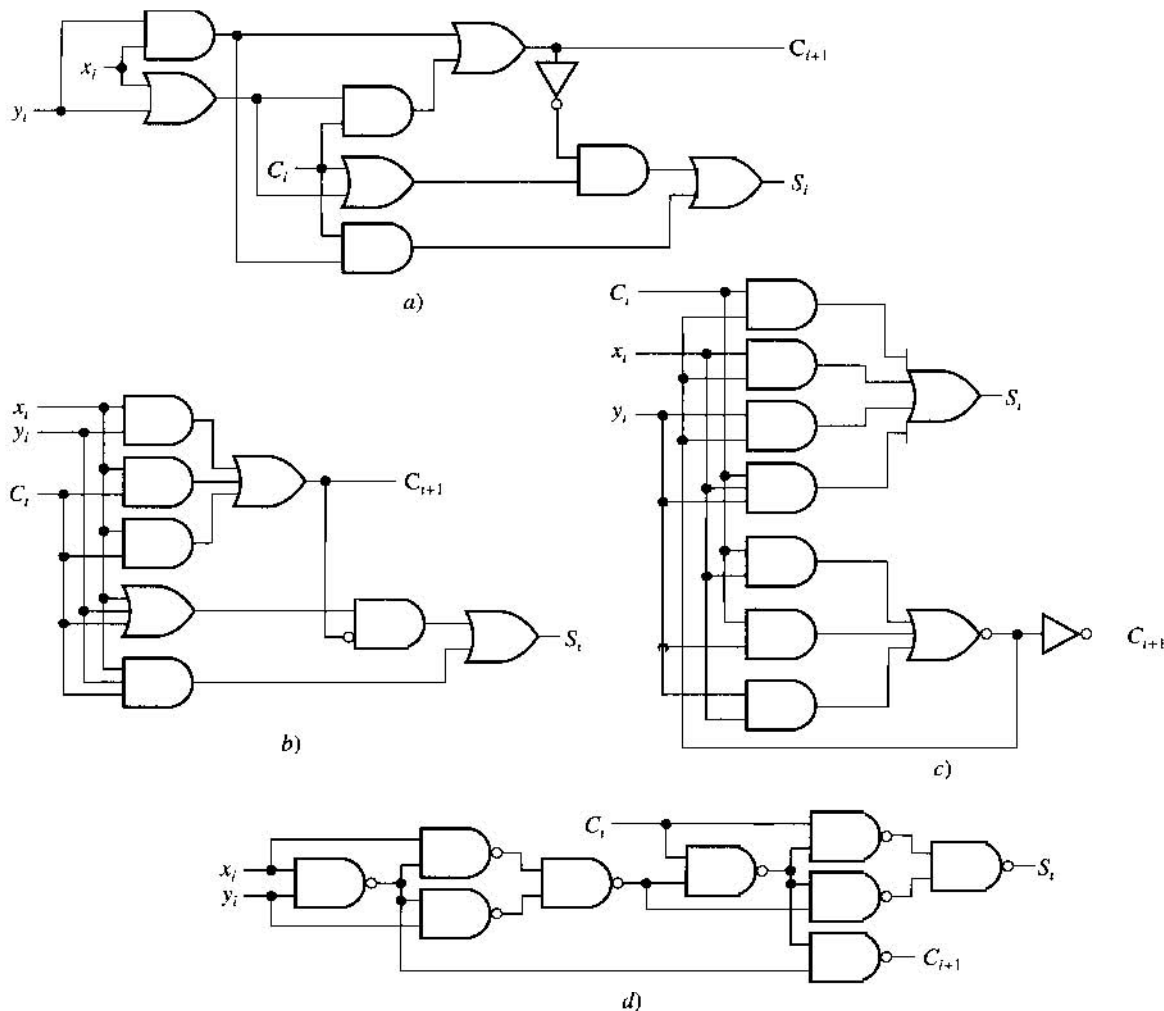


Figura P1

13. Un circuito aceptará dos números binarios de 2 bits x_1x_0 e y_1y_0 y emitirá los productos como números binarios de 4 bits $z_3z_2z_1z_0$. (Si es necesario, revise la multiplicación binaria en el capítulo 1.)
 - a. El resultado se conseguirá (posiblemente) mediante un circuito multinivel con compuertas de dos entradas. Determine expresiones apropiadas para cada salida. ¿Cuántos niveles de compuertas tiene cada salida?
 - b. Diseñe un circuito utilizando un decodificador de 4 a 2^4 líneas con compuertas OR externas.
14. Examine ediciones recientes de libros de datos de fabricantes.
 - a. ¿Cuál es el valor de n en los decodificadores de n a 2^n líneas más grandes?
 - b. Indique cuáles son los tamaños estándar de las ROM.
 - c. ¿Cuáles son algunas dimensiones representativas de un chip PLA?
 - d. ¿Cuáles son algunas de las dimensiones representativas de un PAL?
 - e. ¿Hay un sumador BCD en un CI MSI sencillo?
15. Una función de conmutación de n variables se va a implementar mediante un decodificador de n a 2^n líneas seguido por una compuerta OR externa. La compuerta física que se dispone para este propósito tiene salida tanto OR como NOR. (Es una compuerta ECL.) Por razones prácticas (para evitar problemas del factor de carga de entrada), sería mejor tratar de reducir el número de entradas a una compuerta externa.

- a. Describa cómo implementar la función utilizando la compuerta física disponible si el número de minitérminos contenido en la función es mayor que $2^{n-1} = 2^n/2$.
- h. Ejemplifique con la siguiente función:

$$f = \Sigma(0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 13, 14, 15)$$

16.
 - a. Diseñe un decodificador BCD a decimal utilizando el número mínimo de compuertas AND de dos entradas.
 - b. Repita, utilizando dos decodificadores de 2 a 4 líneas y algunas compuestas AND de interconexión.
17.
 - a. Utilice dos decodificadores idénticos de n a 2^n líneas con entradas habilitadoras para construir un decodificador de $(n + 1)$ a 2^n líneas sin habilitador. Muestre cómo se obtienen las salidas.
 - b. Ejemplifique con dos decodificadores de 2 a 4 líneas.
18. Diseñe un codificador de octal a binario. Éste es un circuito con 8 entradas, x_i , y tres salidas, z_j . Sólo una de las salidas es 1 en cualquier momento. El dígito octal k se representa mediante $x_k = 1$.
19. Diseñe un convertidor de código de dígitos decimales de un código 2 de 5 a siete segmentos (?). Se van a explorar varias posibilidades diferentes, suponiendo que sólo palabras de código válidas se presentarán como entradas.
 - a. Dibuje un diagrama de circuito utilizando un diseño completo de decodificador de 5×7 .
 - b. Suponiendo un diseño que utiliza compuertas discretas:
 - i. Dibuje un circuito para el diseño de suma de minitérminos. (Éste constituiría un decodificador parcial.)
 - ii. Las compuertas AND en el diseño precedente son de cinco entradas. ¿Es posible usar la misma estructura para con compuertas de dos entradas? Justifique su respuesta.
 - iii. Efectúe un diseño mínimo de suma de productos que usa 11 compuertas AND y 7 compuertas OR, cada una con no más de tres entradas.
 - iv. Considere un diseño mínimo de producto de sumas. ¿Éste es más económico que el diseño mínimo de suma de productos?
 - v. Suponga ahora que, además de las palabras de código válidas, también pueden ocurrir palabras inválidas. Modifique lo mejor de los diseños anteriores de manera que cada que haya una palabra de código inválida, se exhiba el símbolo E (correspondiente a error).
20. El convertidor de código del problema 19 se va a diseñar con una ROM. La ROM disponible de tamaño más cercano es una de $2^5 \times 8$. Construya la tabla de programación requerida. Especifique el número de enlaces.
21. El convertidor de código en el problema 19 se va a implementar con un PLA. Se cuenta con un PLA de 5×8 con 12 compuertas AND. Dibuje un diagrama de programación para implementar el convertidor de código que se desea. Especifique el número de enlaces.
22.
 - a. Suponga que el circuito del problema 13 se va a implementar con una PROM de $2^4 \times 4$. Muestre la tabla de programación y dibuje un diagrama apropiado.
 - b. Suponga ahora que el circuito se va a implementar mediante un PLA de 4×4 con 10 compuertas AND. Muestre el diagrama de programación (en la forma de la figura 23 del texto). Compare el número de enlaces con aquéllos de implementación PROM. Construya la tabla de programación en la forma de la figura 25 del texto.
 - c. Suponga ahora que el circuito se va a implementar mediante un PAL. Elabore la tabla de programación en la forma de la figura 25 del texto.
23. Se va a diseñar un circuito combinatorio que tiene tres entradas y seis salidas. La palabra de salida va a ser el cuadrado de la palabra de entrada.
 - a. Diseñe el circuito utilizando una ROM que tenga las dimensiones más pequeñas posible. Elabore la tabla de verdad y especifique el número de enlaces.
 - h. Diseñe el circuito utilizando un PLA con el menor número de términos producto. Elabore el diagrama de programación y especifique el número de enlaces.
24. Los diagramas de programación para dos PLA se muestran en la figura P24.

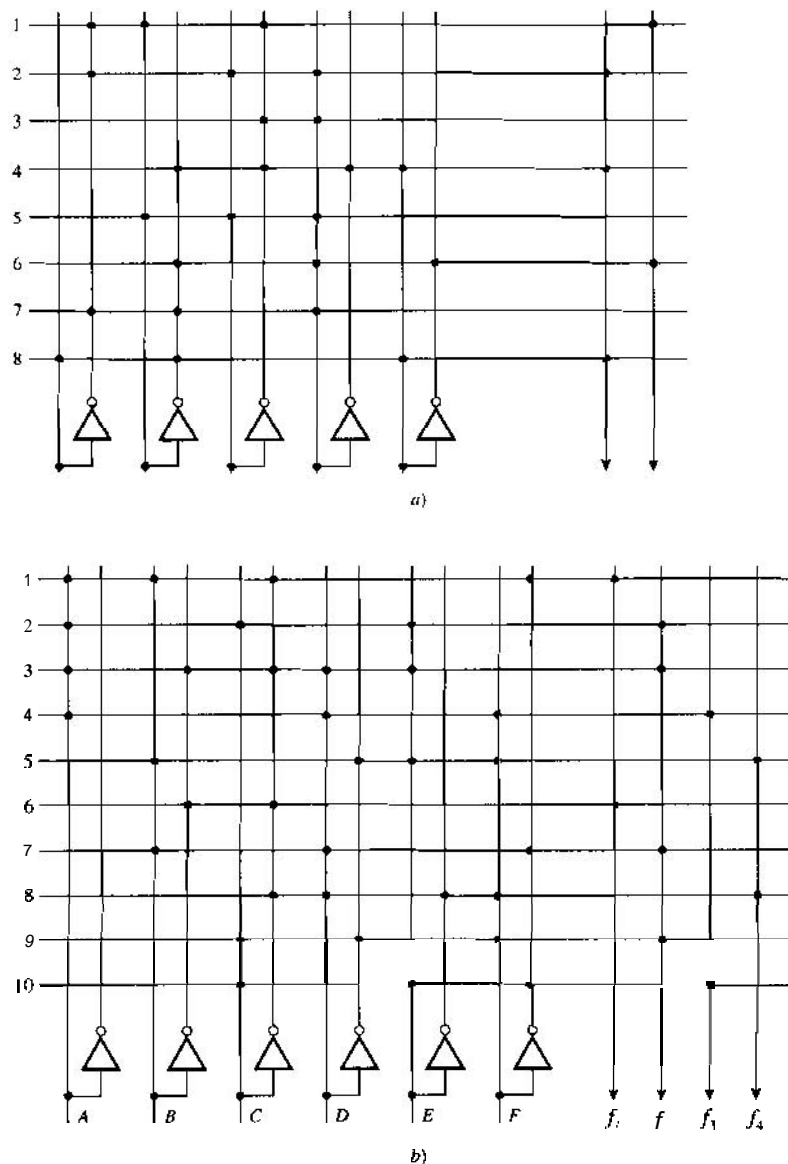


Figura P24

- a. Escriba las ecuaciones de las salidas realizadas por cada PLA. Especifique el número de enlaces.
 - b. Las mismas funciones se van a realizar con una ROM. Especifique las dimensiones de esta misma y el número de enlaces. Integre su tabla de programación.
 - c. Las mismas funciones se van a realizar con un PAL. ¿Es posible hacerlo de ese modo? Si es así, elabore la tabla de programación y especifique el número de enlaces. Si no es posible, explique por qué.
25. (Si lo requiere, revise el capítulo 1 acerca de los códigos de **Hamming**.) Utilizando un decodificador de n a 2^n (para una n apropiada) y cualquier lógica adicional:
- a. Diseñe la lógica de corrección de errores para un código de **Hamming** de corrección de error simple suponiendo 3 bits de mensaje en cada palabra de código. Las salidas del circuito deberán ser:
 - E, indicando que se ha detectado un error.

- IV, indicando que la salida MSG es inválida (evidentemente, IV es 0 cuando ningún error, o sólo un error simple, ha ocurrido).
 - MSG, una salida de 3 bits que contiene el mensaje transmitido corregido en los casos de error y un error.
- b. Diseñe la lógica de corrección de error simple y de *detección de error doble* (SEC-DED) para un código de Hamming de corrección de errores extendido mediante la suma de un bit de paridad por todas las posiciones (esto es, mensaje y paridad). Suponga 3 bits de mensaje en cada palabra de código. Las señales de salida y sus significados serán las mismas que en la parte a
26. Explique en palabras el comportamiento del diagrama de la figura P26. (Las flechas abiertas representan entradas y salidas de múltiples bits.)

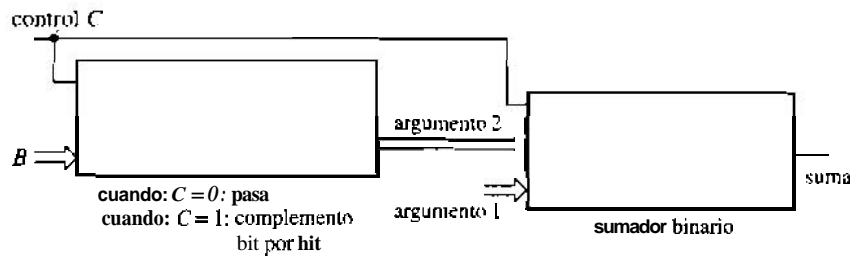


Figura P26

27. Un microprocesador (μp) produce una salida de tres señales de control que tienen el significado indicado en la siguiente tabla. (No se necesita ningún conocimiento de μp para resolver este problema.)

R'	W'	MT'	O'	
0	1	1		el μp quiere leer la memoria
1	0	1		el μp quiere escribir en la memoria
0	1	0		el μp quiere leer un dispositivo de entrada/salida
1	0	0		el μp quiere escribir un dispositivo de entrada/salida
1	1	x		el μp no quiere ninguna de las operaciones anteriores

- a. Diseñe un circuito lógico utilizando un multiplexor adecuado y lógica adicional mínima para transformar estas tres señales en las siguientes cuatro señales, cada una representando una operación:
- $(MR)'$, $(MW)'$, $(IOR)'$, $(IOW)'$
- Cuando se desee (o no se desee) cualquiera de las operaciones, el valor de la señal correspondiente será 0 (1).
- b. Diseñe una implementación de multiplexor para efectuar la transformación inversa.
28. La unidad de anticipación de 4 bits que se muestra en la figura P28a recibe variables generadas y propagadas de las unidades 0 a la 3 constituyendo un grupo similar. También recibe C , la entrada de acarreo a la unidad 0 del grupo. Calcule C_0 , C_1 y C_2 , que son las salidas de acarreo de las unidades 0, 1 y 2, respectivamente. Calcule además las variables generada y propagada, G y P , para todo el grupo. Las salidas de acarreo se generan en paralelo, no en el modo de propagación en cascada.
- a. Obtenga ecuaciones para todas las salidas, y muestre la implementación.
- b. Utilizando unidades de anticipación de 4 bits de tipo anterior y sumadores de 4 bits del tipo que se muestra en la figura P28b, dibuje el diagrama lógico para un sumador de 48 bits utilizando un nivel de anticipación. (Las flechas abiertas representan entradas y salidas de múltiples bits —en este caso, 4 bits, A , por ejemplo, representando un vector de 4 bits: A_0, A_1, A_2, A_3 .)

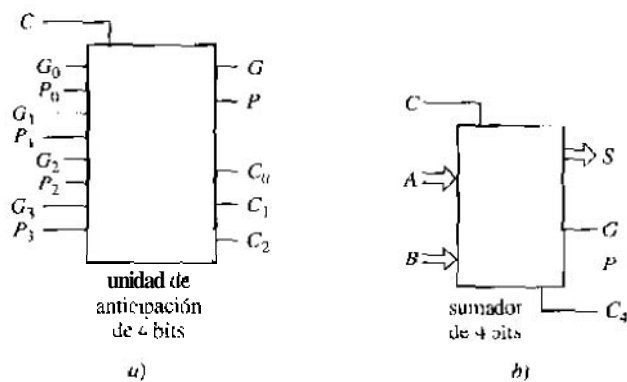


Figura P28

- c. Repita la parte *b* utilizando dos niveles de anticipación, en los que las salidas *G* y *P* de las unidades de anticipación del primer nivel alimentan las entradas *G_i* y *P_i* de las unidades de anticipación de segundo nivel. Compare respecto a la velocidad con el diseño de la parte *b*.
29. Este problema tiene que ver con el diseño de un restador de anticipación de 4 bits (figura P29). El vector de 4 bits *B* (*B₃B₂B₁B₀*) se restará del vector de 4 bits *A*. La entrada de préstamo *C₀* es 1 si y sólo si la siguiente unidad más baja pide un 1 de esta unidad. El vector *D* de 4 bits es la salida de la diferencia, y *C₄* es la salida del que se lleva. *G* y *P* son variables generadas y propagadas de la unidad completa.

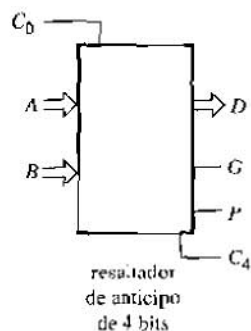


Figura P29

- Proporcione una expresión para cada salida y muestre implementación.
 - Como en el problema 28, existirá más de una manera de definir la variable propagada. Proporcione estas definiciones y compare las diferencias en su implementación.
 - Suponga que se va a efectuar resta de múltiples bits. Con este fin, ¿es posible utilizar unidades de anticipación de 4 bits, del tipo que se describe en el problema 28 en el contexto de la suma, con restadores de anticipación de 4 bits del tipo definido aquí? Justifique su respuesta.
 - Utilizando restadores de 3 bits del tipo descrito en este problema, y también unidades de anticipación de 4 bits adecuadas, diseñe un restador de anticipación de 24 bits.
30. Un codificador de prioridad de 8 entradas (figura P30) tiene ocho entradas solicitadas: *I*(7...0). Un 1 lógico en cualquiera de estas líneas denota la presencia de una solicitud de la fuente correspondiente en cuanto a cierto servicio. La prioridad varía desde la más alta para el 7 hasta la más baja para el 0. La salida *SL* (solicitud local) es 1 si y sólo si hay al menos una solicitud entre las ocho entradas *I*. Si *EH* (entrada habilitada) es 1, el codificador identifica la solicitud que tiene la prioridad más alta y genera como salida su dirección de 3 bits en *A*(0...2). Si ninguna solicitud está activa, produce una salida de dirección cero. Si el codificador no está habilitado (*EH* = 0), produce ceros de salida en *A*. *EH*

(salida habilitada) es 1 si y sólo si el codificador está habilitado ($EH = 1$) y no hay solicitud entre las ocho entradas I .

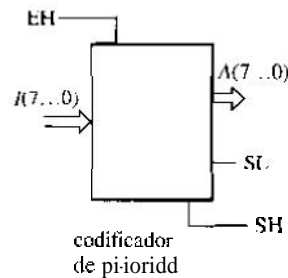


Figura P30

- Obtenga expresiones para cada salida y simplifique.
 - Diseñe un codificador de prioridad de 48 entradas utilizando codificadores de prioridad de 8 bits del tipo descrito en este problema y lógica adicional mínima. Use una configuración repetida.
 - Considerando las señales habilitadoras, EH y SH, como los equivalentes de las señales de acarreo, obtenga expresiones para las variables generada y propagada relativas al codificador de prioridad de ocho entradas. Como en el problema 29, proporcione dos expresiones para la variable propagada y escoja la "mejor". ¿Esto requiere lógica adicional para calcular las variables generada y propagada, o éstas se consiguen de las salidas del codificador de prioridad de ocho entradas descrito aquí?
 - Utilizando unidades de anticipación de 4 bits adecuadas, diseñe una implementación de anticipación para un codificador de prioridad de 48 entradas y compare su velocidad con el diseño de la parir h .
 - Suponga que el codificador de prioridad de ocho entradas tiene señales de deshabilitación. El Y y SI , en vez de las señales de habilitación EH y SH. Repita las partes c y d considerando las señales de deshabilitación como el equivalente de señales de acarreo.
31. Un decodificador BCD a siete segmentos tiene señales "de limpieza", BI y BO, para ayudar a suprimir los primeros 0 a la izquierda para exhibidores de enteros y los últimos 0 a la derecha para exhibidores de fracciones. Cuando BI es 1, si el dígito de entrada es 0, todas las salidas deben ser 0; esto es, el dígito será eliminado. Cuando BI es 0, no habrá eliminación, sino que en ese caso BO es una señal de limpieza para el siguiente dígito. En la figura P31a se muestra un diagrama.

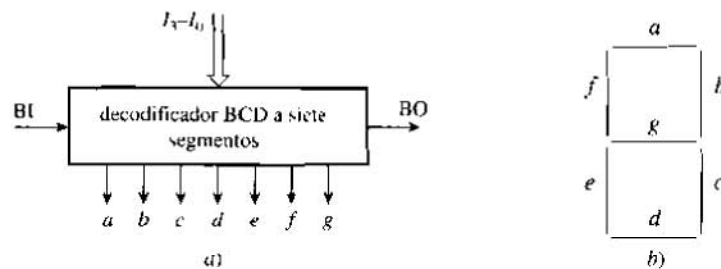


Figura P31

- Proporcione expresiones para las salidas BO, a y f.
- Diseñe un exhibidor de 8 bits cada uno con cuatro dígitos para las partes enteras y fraccionarias. El dígito entero menos significativo nunca debe ser eliminado, incluso si la parte entera del número es 0.
- Considerando los tiempos lentos de respuesta humana, la implementación mediante repetición en la parte b debe ser adecuada. Sin embargo, por propósitos pedagógicos, suponga que desea diseñar una implementación con anticipación del exhibidor, de manera que cada dígito se establezca con mayor rapidez en el estado de eliminación o en el de no eliminación. Al considerar BI y BO

como señales de acarreo, proporcione expresiones para las variables generada y propagada de este decodificador.

- d. Suponga que, en lugar de las patillas B1 y B0, el decodificador tiene patillas DB1 (entrada "no en blanco") y DB0 (salida "no en blanco). En este caso considere éstos como señales de acarreo y repita la parte c.
32. Demuestre formalmente que si la variable propagada P_i para el sumador anticipado se define como la suma booleana de A_i y B_i en lugar de su OR exclusiva, las salidas de la suma y el acarreo del sumador se seguirán calculando de manera correcta. También ofrezca una demostración informal. ¿Cuál definición es mejor para fines de implementación?
 33. Un selector de datos de 4 bits tiene cuatro entradas de datos, D_3, \dots, D_0 y dos entradas de selección, s_1, s_0 . La salida z es una de las entradas de datos que se selecciona mediante las entradas de selección. De tal modo, $z = D_2$ cuando $s_1 s_0 = 10$.
 - a. Dibuje un diagrama AND-OR del selector de datos.
 - b. Otro circuito consiste en dos compuertas XOR. Las entradas a XOR1 son dos señales A y B . Las entradas a XOR2 son la salida de XOR1 y una tercera señal C . Dibuje este circuito y escriba su salida en términos de A , B y C .
 - c. Elija las entradas de selección y las entradas de datos en la parte a en términos de A , B y C de manera que los circuitos en las partes A y B tendrán las mismas salidas. Si hay más de una elección, muéstrelas todas.
 34.
 - a. Diseñe un sumador BCD utilizando una ROM (y cualquier otra lógica necesaria), considerando que sólo se usan como entrada palabras BCD legales. Especifique las dimensiones de la ROM y muestre un diagrama esquemático.
 - b. Describa la tabla de programación e ilústrela (al menos parcialmente.)
 - c. Especifique el número de enlaces.

Ca

Capítulo 5

Componentes de circuitos secuenciales

Las salidas en cualquier tiempo determinado dependen sólo de las entradas precisamente en ese tiempo, no de la historia de las entradas pasadas. Esto implica:

- **Que las compuertas en el circuito responden a las entradas sin retardo, o**
- **Que los intervalos de tiempo entre entradas sucesivas son tan largos comparados con el tiempo de respuesta de las compuertas, de modo que las respuestas de salida para un conjunto de entradas pasadas haya ya ocurrido antes de que se produzca la siguiente entrada.**

Existen situaciones, sin embargo, en las que una salida depende no únicamente de las entradas presentes sino también del estado del circuito en el momento en que llegan esas entradas. El estado del circuito en cualquier tiempo dado, a su vez, depende de la historia de las entradas pasadas. Esto quiere decir que debe haber un mecanismo para almacenar la información transmitida por la secuencia de entradas pasadas. Este capítulo trata de los dispositivos que almacenan entradas pasadas —ya sea una entrada pasada simple o una secuencia de ellas.

1 DEFINICIONES Y CONCEPTOS BÁSICOS

El almacenamiento de información en relación con entradas pasadas implica la memorización de dichas entradas pasadas. En este capítulo presentaremos varios circuitos específicos que sirven como celdas (llamadas a menudo *celdas primitivas*) para el almacenamiento de una entrada pasada simple. La interconexión de estas celdas de memoria, junto con los circuitos combinatorios, puede tener que ver con situaciones donde una salida depende de la entrada presente y de una cadena finita de entradas pasadas. Esta explicación da origen a la siguiente definición:

*Un circuito digital es secuencial si sus salidas en cualquier momento determinado son funciones tanto de las entradas externas en ese momento como de las secuencias de entradas pasadas.*¹

Con esta liase, es posible construir modelos de un circuito secuencial, como se indica en la figura 1.

Hay dos tipos de entradas al bloque de lógica combinatoria del circuito secuencial. Hay entradas provenientes del mundo externo, denominadas *primarias*. También existen entradas secundarias que describen la condición (o estado) en el cual se encuentra el circuito a la llegada de las entradas presentes. Estas entradas secundarias se retroalimentan desde la memoria. Am-

¹ Una variación de este caso genera se explicará en el capítulo 7.

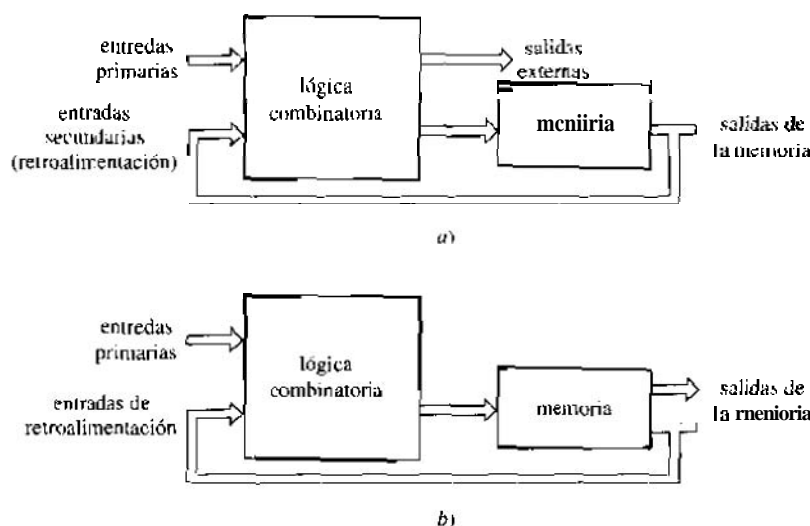


Figura 1. Modelos de un circuito secuencial. a) Modelo de Mealy. b) Modelo de Moore.

Los conjuntos de *entradas* se procesan mediante lógica combinatoria. En el modelo conocido como *máquina de Mealy*, hay dos tipos de *salidas* del bloque de lógica combinatoria: salidas hacia el exterior y salidas que *representan* nueva información que se almacena en la memoria. La memoria incorpora la nueva información, generada por las últimas entradas, a la información previamente almacenada. Las *salidas de memoria* se retroalimentan como *entradas secundarias* a la lógica combinatoria.

El proceso de modificación de los *contenidos de la unidad de memoria* recibe el nombre de *escritura en memoria*. El proceso inverso — adquirir como una salida la información actualmente almacenada en una unidad de memoria — se llama *lectura de memoria*.

En el modelo de Mealy que se muestra en la figura 1a, las salidas externas dependen tanto de las entradas del mundo exterior como de las *entradas de retroalimentación de la memoria*. Hay, sin embargo, circuitos digitales en los cuales la salida *no depende* de manera directa de las entradas externas. En vez de eso, las *entradas externas provocan cambios en la memoria*, después de lo cual se emiten salidas *externas* desde ella. Como en la *máquina de Mealy*, otro conjunto de *salidas de la memoria* se convierte en *entradas de retroalimentación para la lógica combinatoria*. El modelo de un circuito secuencial de estas características se ilustra en la figura 1b y se conoce como *máquina de Moore*. En capítulos subsecuentes abordaremos ambos modelos.

Las dificultades de temporización son inherentes en los circuitos de conmutación. Con la variabilidad tanto en el tiempo de llegada de las entradas como en el *retardo de tiempo inherente* a las compuertas, quizá resulte difícil seguir el rastro a lo que es una entrada pasada y a lo que es una *presente*. Como ejemplo, suponga que una *entrada variable cambia su valor en algún tiempo*. Después de cierto *retardo de propagación*, la respuesta a este cambio se percibirá tanto en la entrada a la memoria como en las terminales de salida. Por consiguiente, cambiará el *contenido de la memoria*. Puesto que hay retroalimentación hacia la entrada en ambos modelos este *cambio* tal vez ocasione una *modificación adicional en la salida*. A partir del tiempo del *cambio inicial en la entrada* y el *cambio final en la salida* ocurre un periodo de inestabilidad. Si un cambio adicional en la *entrada primaria* ocurre durante este *periodo de inestabilidad*, el *circuito podría fallar en dar una respuesta confiable* debido a la *confusión en los contenidos de la memoria* cuando arribó la *segunda entrada*. (Existirá una *señal espuria*.)

Un método común para superar este problema radica en introducir una señal adicional en la forma de un tren de *rindas de pulso periódicas denominado reloj*, como se muestra en la figura

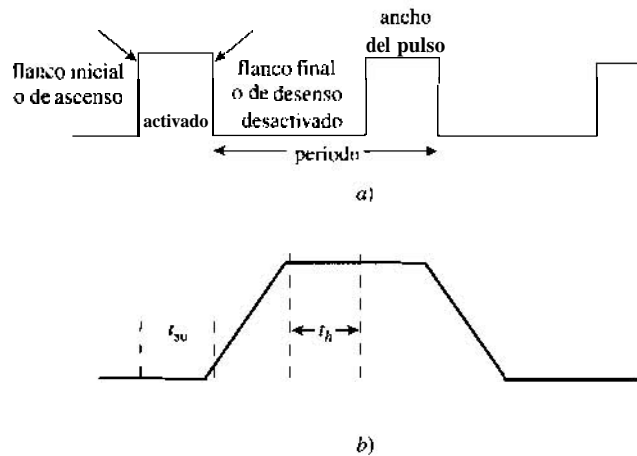


Figura 2. a) Un tren de pulsos periódico. b) Un pulso de reloj "realista"

2a. (Esta señal de reloj se genera mediante lo que se conoce como *oscilador*, por lo general de cristal de cuarzo.) La parte de memoria del circuito en la figura 1 se construye de modo tal que está **deshabilitada durante el tiempo que el pulso de reloj está ausente**; la actividad en el circuito se produce **sólo en la presencia del pulso de reloj**. Si éste es suficientemente estrecho, es improbable que ocurra más de un cambio de entrada durante el tiempo en que el pulso de reloj está **activado**.² En realidad, en muchos circuitos, el inicio del pulso de reloj (el flanco de ascenso) habilitará al circuito se trata de la llamada activación por flanco, que se describirá en la última sección. Ninguna actividad subsecuente se producirá en el circuito, no importa **cuál sea el ancho** del pulso de reloj, hasta **el inicio del siguiente pulso de reloj**. (La **activación por flanco** también puede hacerse mediante el flanco de descenso del pulso de reloj.) El tipo de circuito que **acaba de describirse** se conoce como **circuito secuencial con reloj**. Es posible interconectar varios circuitos secuenciales con reloj, aunque sería muy confuso si fueran diferentes los relojes que habilitan a los circuitos, **respectivos**. Si todos los relojes **habilitadores** en un circuito **interconectado** de este tipo son iguales, entonces la actividad en todos los circuitos ocurrirá **de manera síncrona**. El resultado recibe el nombre de **circuito secuencial síncrono**, debido a que todas las partes del circuito están sincronizadas por el mismo reloj del sistema.

Un periodo de la forma de onda del reloj incluye un intervalo **de tiempo en el que el pulso del reloj es 1** ("alto" en lógica positiva, alguno diría "activo en nivel alto") y otro intervalo de tiempo cuando éste es 0 ("bajo" en lógica positiva). Si son **iguales estos dos intervalos**, la señal es una onda cuadrada. (Resulta difícil concebir una onda **cuadrada como una** secuencia de pulsos.) Si el intervalo alto es mucho más corto que el intervalo bajo, la **señal** corresponde a un tren de pulsos que se hacen positivos. Por otro lado, si el intervalo de tiempo en estado alto es mucho más **largo** que el intervalo bajo, la **señal** es un tren de pulsos que se hacen negativos. (Para tener una **percepción de lo anterior**, antes de continuar, dibuje una forma de onda en la que la duración del intervalo bajo sea 10 veces la **correspondiente al intervalo alto** y viceversa.)

Los rasgos importantes de la señal de reloj son los siguientes:

- Ciclo de trabajo.
- Frecuencia de reloj.
- La pendiente de los flancos.
- Estabilidad de la frecuencia y la forma de onda.

² La figura 2 es apropiada para el sistema "activo en alto" utilizado aquí. para el sistema "activo en bajo", los pulsos iniciarían con una transición negativa.

El *ciclo de trabajo* es la fracción del periodo de reloj en el que la señal de reloj es alta. En el caso de una onda cuadrada, el ciclo de trabajo es igual a 0.5; es común un ciclo de trabajo menor que 0.01 (1%). La frecuencia es particularmente importante. Puesto que la temporización de toda la actividad en un circuito con reloj se basa en la ocurrencia de un pulso de reloj, la rapidez con la cual el circuito efectúa sus operaciones depende de la frecuencia. En las operaciones de computadora se consideran bajas las frecuencias del orden de 1 MHz (un millón de operaciones por segundo); es común el valor de 5 a 50 MHz. Los microprocesadores de hoy día presentan frecuencia de reloj tan altas como 600 MHz.

Las dos últimas características de la señal de reloj dependen de la calidad y el diseño del oscilador. Aunque éstas son importantes, los diseñadores de lógica no tienen control sobre este tipo de características; deben aceptar lo que proporcionan los diseñadores del oscilador.

En una situación ideal, los lados del pulso de reloj aumentan y disminuyen en un tiempo nulo. Sin embargo, la vida real se diferencia de la ideal en dos formas: a) los lados aumentan y caen en un tiempo distinto de cero, y b) las pendientes de los lados no cambian de cero a un valor distinto de él (o viceversa) en un tiempo cero; b) da lugar a pulsos que son trapezoidales en vez de rectangulares, como en la figura 2b. De manera más realista, las pendientes de las esquinas tanto de ascenso como de descenso del pulso de reloj cambian de modo gradual, por lo que las "esquinas" son curvas. (Véase la figura 22 del capítulo 2.) Ignoraremos esta curvatura en el reloj y otras señales en la explicación que sigue.

Ejercicio 1

- Estime el valor aproximado del ciclo de trabajo en la figura 2a.
- Suponga que el ciclo de trabajo de un tren de pulsos es 1% y que su frecuencia es 10 MHz. ¿Cuánto dura un periodo en nanosegundos? Suponga que un periodo ocupa 4 cm sobre un eje horizontal; dibuje a escala un tren de pulsos de estas características. ¿Cuánto dura en nanosegundos el pulso de reloj? ¿Cuántos centímetros mide el ancho del pulso de reloj?

En este capítulo, se presenta un número considerable de circuitos que sirven como elementos de celdas de memoria, empezando con el más simple. Cada nuevo circuito que se introduzca se justificará señalando las deficiencias del anterior e indicando cómo el nuevo circuito lo mejora o tiene una característica útil de la que los anteriores carecen. Por la naturaleza de este material, el capítulo es bastante descriptivo, y también encontraremos un poco de matemáticas y análisis. El propósito aquí no es facultar para que se embarque en el diseño de tales dispositivos, sino ayudarlo para que aprenda lo suficiente de sus características y las use con facilidad en el diseño de sistemas más grandes.

2 CERROJOS Y FLIP-FLOPS

Una de las necesidades en un circuito secuencial es el almacenamiento de información en memoria respecto a la condición presente del circuito como resultado de entradas pasadas. Un dispositivo *biestable*, que puede encontrarse en una de dos condiciones estables, puede almacenar un bit de información. Un interruptor de una lámpara, por ejemplo, estará en cualquiera de dos posiciones. La información relativa a que la luz esté activada o desactivada, está contenida en la posición del interruptor. En circuitos digitales, los elementos más comunes de memoria son dispositivos electrónicos conocidos como *flip-flops*. Consideraremos varias variedades de ellos, cada una útil en ciertas circunstancias.

El tema se presentará utilizando el circuito de la figura 3a, el cual muestra 2 inversores de acoplamiento cruzado. Advierta que la interconexión de apariencia compleja en la figura 3a puede volverse a dibujar como en la figura 3b. ("Desenrolle" el circuito anterior y confírmelo.) La entrada de cada inversor es la salida del otro, sin ninguna entrada externa. Éste es un ejemplo

Cerrojos S

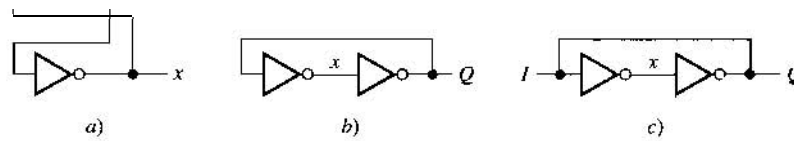


Figura 3. Un circuito de inversores biestable.

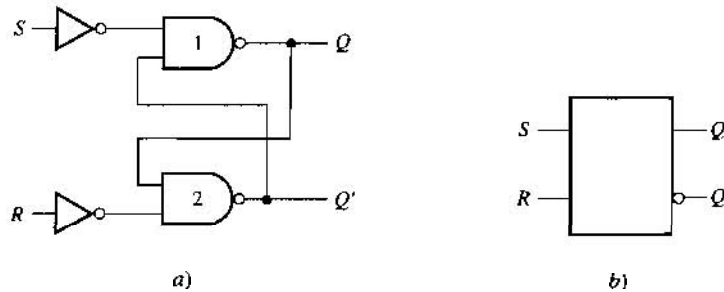


Figura 4. Diseño de un cerrojo SR. a) Con NAND. b) Símbolo esquemático.

digital de un dispositivo biestable. Para confirmar esta afirmación, suponga que la salida superior Q en la figura 3a tiene un valor lógico 1. Puesto que ésta es la entrada para el inversor de abajo, el último tendrá la salida $x = 0$. Pero x es también la entrada al inversor superior, lo que confirma la salida supuesta $Q = 1$.

(Efectúe usted ahora la siguiente tarea: suponga que $x = 1$ y confirme que $Q = x' = 0$, verificando que $x = (x')' = 1$, como se supuso.) De tal modo, la salida de este circuito puede permanecer estable en cualesquiera de los valores $Q = 0$ o $Q = 1$. De ahí proviene la terminología *biestable*. El valor que la salida toma en realidad depende de una posible entrada externa, como se muestra en la figura 3c. Cada una de estas dos salidas de inversor es el complemento de la otra.

Cerrojos SR

El circuito biestable en la figura 3c tiene dos fuentes de datos para la entrada al primer inversor: la entrada externa y la retroalimentación desde la salida del segundo inversor. Resulta difícil escribir un nuevo valor en el circuito debido a que la retroalimentación se diseña para mantener el valor existente. La retroalimentación y la entrada externa pelean por determinar el valor almacenado en el elemento de memoria. La figura 4a muestra el diseño de un elemento de memoria que recibe el nombre de *cerrojo SR*. El diseño recurre a dos compuertas NAND precedidas por inversores.³ (Analice en un ejercicio otro diseño con dos compuertas NOR.)

Lo que ocurre en este circuito sin los inversores. La ventaja del circuito de la figura 4a comparado con el de la figura 3c es que las entradas externas pueden influir en el estado de la memoria sin pelear con la retroalimentación en el circuito.

La característica novedosa, diferente de todo lo que usted ha visto en circuitos combinacionales pero que se encuentra en el circuito de la figura 4a, es el acoplamiento cruzado, o retroalimentación, de la salida de cada compuerta hasta una entrada de otra compuerta. Las dos entradas externas (primarias) reciben el nombre de *establecimiento (S)* y *reestablecimiento (R)*.

No habría inversores si se usara lógica negativa o si las entradas S y R fueran activas en nivel bajo en la lógica mezclada; véase el capítulo 2

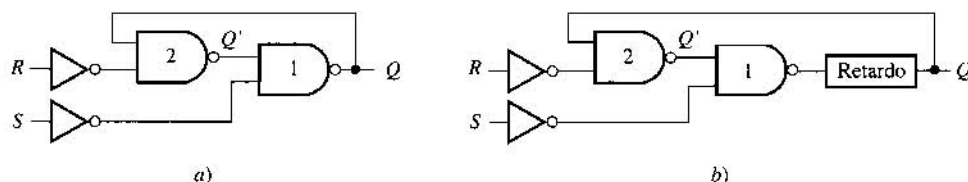


Figura 5. Cerrojo SR con retroalimentación y retardo enfatizado.

Con base en la experiencia relativa a dispositivos de acoplamiento cruzado de la figura 3, suponemos que cada salida de compuerta NAND es el complemento de la otra. Estas salidas se designan universalmente Q . Sin embargo, cuando se realizan conexiones externas de una salida de celda de memoria a otros dispositivos, es posible dar otras designaciones a estas variables externas. A menudo las usaremos y con este fin, incluso cuando expliquemos una celda de memoria simple. En la figura 4h se muestra un diagrama esquemático. La burbuja en la salida inferior implica que esta señal es el complemento de la denominada Q . Si se muestra la burbuja, en realidad no es necesario proporcionar cualquier otra indicación, tal como la Q' que se muestra en la figura 4h. Se podría incluso mal interpretar esto como una doble negación."

Se suponen reales las compuertas físicas que se muestran en la figura 4, no simplemente representaciones gráficas de la operación booleana NAND. En un principio podría parecer sorprendente que una mera interconexión de un par de compuertas pueda conducir a otra cosa que no sea un circuito estrictamente combinatorio. La respuesta radica a) en la retroalimentación proporcionada por el acoplamiento cruzado, y b) en el retardo de las señales al recorrer las compuertas reales.

Es posible subrayar la característica de retroalimentación redibujando el cerrojo de la manera que se indica en la figura 5a.

A pesar de que existe un retardo de propagación a través de cada compuerta física real, vamos a suponer por el momento —como una *digresión sólo en este párrafo*— que todos los retardos del circuito de la figura 5a se agrupan en un lugar, como se indica en la figura 5b; las compuertas en el último circuito se consideran ideales y, por tanto, responden de manera instantánea. Un modelo con tales características para tratar los retardos de propagación de compuertas ofrecería resultados bastante precisos. El efecto de un cambio en la entrada S o R se percibe de inmediato en la salida marcada q (para distinguirla de la salida final, Q) en la figura 5b. No es hasta después de un retardo, sin embargo, que este evento llega a Q y se retroalimenta a la entrada.

En explicaciones subsecuentes, el bloque de retardo en la figura 5b no se mostrará de manera explícita. Todas las compuertas se considerarán físicas y reales; así, la presencia de retardo de propagación en cualquier compuerta se supondrá implícitamente.

Donde exista diferencia, el análisis tomará en cuenta este retardo. Cuando el cerrojo SR no está excitado (esto es, tanto S como R son iguales a 0), éste se comporta como un dispositivo biestable, capaz de mantener la salida Q en 1 o en 0 de manera indefinida, de igual modo que los inversores de acoplamiento cruzado de la figura 3. Cambiar la salida del cerrojo requiere que la señal de entrada S o R se vuelva 1. La condición del cerrojo, indicando si la salida es 0 o 1, se denomina su *estado*. (Esta terminología se extenderá después a circuitos que contienen cualquier número de elementos de almacenamiento: el estado del circuito se referirá a la colección de valores de salida de todos los dispositivos de almacenamiento de memoria.)

⁴ Consulte varios libros de diseño lógico y advierta que algunos de ellos ponen las literales Q y Q' dentro del rectángulo que representa al dispositivo. ¿Qué piensa usted de la propiedad de utilizar una literal Q' y seguirla con una burbuja?

S	R	Q	Q ⁺
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

(a)

S	R	Q ⁺
0	0	Y
0	1	0
1	0	1
1	1	x

(b)

Figura 6. Tabla de transición para el cerrojo SR.

Ejercicio 2. Uno a la vez, suponga que el cerrojo está en cada uno de los dos estados ($Q = 1$, $Q = 0$). En cada estado, considere entradas $SR = 00$ (ambas entradas 0) y confirme que el cerrojo SR permanecerá en cualquier estado en el cual se encuentre.⁵

Suponga que el cerrojo se encuentra en uno de sus dos estados $Q = 0$ o $Q = 1$. Llamado el *estado presente*. Después de esto las entradas S y/o R cambian en una nueva combinación de valores; luego de un retardo correspondiente al retardo de propagación a través de las compuertas físicas, debe haber una transición a un *nuevo* (pero no necesariamente diferente) estado, denominado *estado siguiente*. Si el tiempo presente se indica mediante t_n , el tiempo en la ocurrencia de una transición se marca como t_{n+1} . La correspondiente secuencia de estados se indicará mediante una de las siguientes notaciones:

$$\begin{aligned} Q(t_n) &\rightarrow Q(t_{n+1}) \\ Q^n &\rightarrow Q^{n+1} \\ Q &\rightarrow Q^+ \end{aligned} \quad (1)$$

La última de éstas es la más simple, y la usaremos con frecuencia.

Lo que necesitamos ahora es trabajar lo que será el estado siguiente del cerrojo SK para cualquier combinación de entradas y cualquier estado presente. Vamos a iniciar con el estado presente $Q = 0$ y $SR = 00$ en la figura 4a. (El ejercicio 2 debe haberlo convencido de que es posible esta combinación de entradas y estado presente.) Supongamos ahora que SR se convierte en 10; esto es, la entrada S cambia en 1.

Una de las entradas de compuerta 1 pasa a ser 0; en consecuencia, Q se vuelve 1. En este caso ambas entradas de la compuerta 2 son 1; así que Q' se vuelve 0. Puesto que ésta es la retroalimentación para la compuerta 1, usted debe determinar si afecta la salida de la compuerta 1. (No lo hace.) Por consiguiente, el estado siguiente es $Q^+ = 1$. La entrada S ha pasado a la salida. Afirmamos que el cerrojo se ha establecido.

Ejercicio 3. Sigue el mismo proceso que acaba de describirse para determinar el estado siguiente Q^+ cuando SR se convierte en 01 a partir del valor precedente de 00. La respuesta se da en la denominada *tabla de transición* que se muestra en la figura 6a. Advierta que cuando $R = 1$ (mientras $S = 0$) la salida se ha transformado en 0; afirmamos que el cerrojo se ha reestablecido.

El caso $SR = 11$ es anómalo. En esta situación, cada una de las dos compuertas NAND tiene una entrada que es 0. Resultaría que tanto Q como Q' se convertirían en 1, ¡un resultado inconsistente! Además, ¿cuál sería la consecuencia si SR subsecuentemente pasa de 11 a 00?

Advierta que la notación SR no pretende ser la AND de las entradas S y R , sino que sólo es una manera sencilla de identificar la secuencia de valores que sigue, esto es, 00. Esta práctica puede resultar confusa: reduciremos la confusión siempre que convenga mostrar de manera explícita la operación AND de la manera usual: $S \cdot R$.

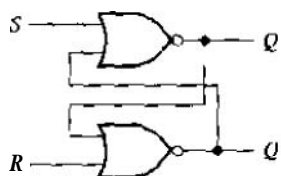


Figura 7. Diseño de compuerta NOR del cerrojo SR

Las entradas para cada compuerta serían 1 y, por consiguiente, ambas salidas se harían 0, otra inconsistencia. Pero, más importante, si las entradas *pudieran* incluso cambiar exactamente al mismo tiempo, puesto que es improbable que los retardos de propagación de las dos compuertas sean exactamente iguales, una de las salidas alcanzaría primero el 0.⁶ Debido a la retroalimentación, la salida de la otra compuerta se convertiría entonces en 1. Así, en algunas circunstancias $Q = 1$ y en otras, $Q = 0$, una incertidumbre que resulta inaceptable. En consecuencia, la combinación de entrada $SR = 11$ no puede tolerarse.

Es posible eliminar la condición $SR = 11$ imponiendo el requerimiento lógico de que $S \cdot R = 0$. Puesto que $SR = 11$ no va a ocurrir, no nos preocupamos de lo que será la salida para una combinación de entradas de este tipo, como se indica en la figura 6. Por consiguiente, la ecuación de transición completa para el cerrojo SR obtenido a partir de la tabla de transición en la Figura 6a es:

$$\begin{aligned} Q(t+1) &= S(t) + R'Q(t), & S(t) \cdot R(t) &= 0 \\ Q^+ &= S + R'Q, & S \cdot R &= 0 \end{aligned} \quad (2)$$

(Confirme este resultado escribiendo una suma de minitérminos, incluyendo valores irrelevantes, y reduciéndola utilizando álgebra booleana. ¿La notación $SR = 0$ en (2) implica de manera inambigua la AND booleana, y no una yuxtaposición de S y R?)

Examinamos ahora la tabla de transición de la figura 6a desde un punto de vista diferente. Cuando ya sea $S = 1$ o $R = 1$, pero no ambas, el estado siguiente toma el valor 0 o 1 *independientemente del estado presente*. Esto es, el estado siguiente depende sólo de las entradas, no del estado presente. Esto significa que, para estas combinaciones de entrada particulares, el circuito se comporta como 1 combinatorio.

Sin embargo, advierta también que para la combinación $SR = 00$, el siguiente estado *depende* del estado presente. Toda esta información en las transiciones se presenta sin ambigüedad en la tabla de transición reducida de la figura 6b.

Ejercicio 4. Una estudiante de lógica digital, al advertir que el diseño del cerrojo SR que utiliza compuertas NAND requiere inversores en las entradas, tiene la siguiente idea: ¿Qué sucedería si las compuertas NAND se sustituyeran por compuertas NOR, pero sin los inversores? Construyó el diagrama de la figura 7. Elabore una tabla de transición para este circuito similar a la de la figura 6. Comparando las dos tablas, confirme que este diseño produce también un cerrojo SR. *

Problemas de temporización y cerrojos SR con reloj

El diseño del cerrojo SR de la figura 4a no incluye un reloj. La inclusión de un reloj como una entrada independiente podría ocasionar que todos los cambios de estado en un circuito se llevaran a cabo de manera simultánea. Ello podría resultar también en que todas las transiciones de estado se efectuaran de manera confiable, sin la incertidumbre de la salida que se describió en la sección 1.

⁶ Las dos compuertas parecen estar compitiendo o "corriendo" una en contra de otra para ver cual de ellas tendrá primero una salida 1. Por esta razón, este caso se conoce como una condición de *carrera*. Este asunto se analizará con mayor detalle en el capítulo 7.

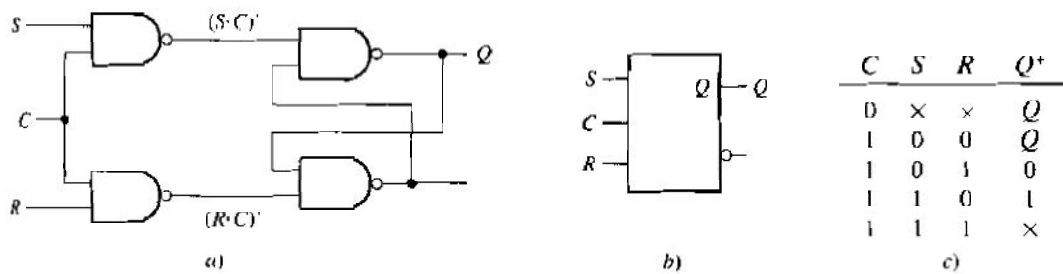


Figura 8. Un cerrojo SR con reloj y su tabla de transición

En la figura 8a se muestra el diseño de un cerrojo SR con reloj utilizando compuertas NAND. Las dos compuertas a la izquierda del cerrojo reciben el nombre de *compuertas de control*. La señal de reloj es una de las dos entradas a cada compuerta guía. Cuando la señal de reloj C es 0, las entradas S y R no tienen influencia en el estado Q . En este caso, el circuito es equivalente a un cerrojo con $S = R = 0$. (Compruébelo.) Cuando la señal de reloj es 1, por otra parte, el comportamiento del circuito se reduce al del cerrojo SR de la figura 4a. Confirme la tabla de transición que se muestra en la figura 8c.

Ejercicio 5

- Utilizando la tabla de transición de la figura 8c, construya un mapa de cuatro variables con C y Q como dos de ellas, y S y R como las otras dos. Las entradas en el mapa serán los estados siguientes. (Complete su propio mapa antes de verificarlo utilizando el mapa terminado que se indica en la respuesta.)
- A partir del mapa, obtenga una expresión mínima para el estado siguiente.

Respuesta

		CQ			
		00	01	11	10
SR	00		1	1	
	01		1		
	11		1	x	x
	10		1	1	1

Advierta que para $C = 0$, la expresión para Q^+ en la respuesta para el ejercicio 5 se reduce a $Q^+ = Q$. Esto nos indica que el nuevo estado es el mismo que el estado presente. Siempre y cuando el reloj permanezca en 0, no ocurrirá ninguna transición de estado. Para $C = 1$, por otro lado Q^+ se reduce a la expresión del estado siguiente para el cerrojo SR en (2), como debe ser. Se dice que el cerrojo será *transparente* cuando $C = 1$, debido a que las salidas responden a cambios en las entradas. Para referencia futura, la expresión correspondiente a Q^+ en el ejercicio 5 se da a continuación.

$$Q^+ = C'Q + CS + R'Q \quad (3)$$

En la figura 8b se presenta un símbolo, o diagrama de bloques, para el cerrojo SR con reloj. Observe que la presencia del reloj no cambia la indeterminación de la salida para la condición $S = R = 1$.

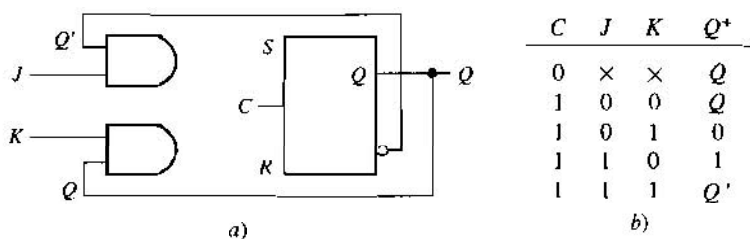


Figura 9. Cerrojo JK.

Cerrojo JK

La operación adecuada del cerrojo SR requiere que **ambas** entradas no sean 1 en forma **simultánea**. Esto constituye un dolor de cabeza y provoca problemas prácticos. Un cambio de diseño que supera esta dificultad sería más que bienvenido. En la **figura 9a** se **muestra un** diseño modificado utilizando dos compuertas AND cuyas salidas **desempeñan** el papel de S y R. Una de **las** dos entradas a cada una de estas compuertas AND se **retroalimenta** a partir de **las** salidas del cerrojo SR. Las otras dos entradas (externas) se marcan, respectivamente, como J y K. Las expresiones **para** S y R son $S = JQ'$ y $R = KQ$. (Confírmelo.) La inserción de éstas en (3) produce la ecuación de transición.

$$\begin{aligned}
 Q^* &= C'Q + C(JQ') + Q(KQ)' \\
 &= C'Q + CJQ' + K'Q \\
 &= JQ' + K'Q \quad \text{por } C = 1
 \end{aligned}
 \tag{4}$$

El uso de las expresiones $S = JQ'$ y $R = KQ$ conduce a $S'R = JKQ'Q = 0$: de este modo, se satisface de manera automática la condición de que S y R nunca sean simultáneamente 1.

Es evidente que el cerrojo SR con reloj no tiene ventaja sobre el JK en circuitos síncronos, por lo que éstos se usan rara vez como dispositivos de memoria en tales circuitos.⁷

Cerrojo maestro-esclavo

El cerrojo JK con **reloj** supera algunos de los problemas de temporización y la prohibición **Contra** **entradas** simultáneamente altas en el cerrojo SR, **aunque** permanecen otros problemas de temporización. Es cierto que la **ocurrencia** de un pulso de reloj inicia una **transición** de estado sobre la base de las señales J y K presentes en ese tiempo. Suponga que tanto J como K son 1 cuando llega un pulso de **reloj**. De acuerdo con (3) o la tabla de transición de la figura 8c, la transición es $Q^* = Q$, y por ello cambia el estado. Después de que se termina la transición, el nuevo estado se retroalimenta a las entradas de las compuertas de control.

Si el retardo en este proceso es relativamente **pequeño** y el pulso de reloj sigue presente, ocurrirá una **transición** adicional de acuerdo con la tabla de transición. Este proceso continuará hasta que el pulso de reloj se haga 0. La salida del cerrojo, por tanto, será incierta, dependiendo del ancho del pulso de reloj relativo al tiempo de propagación a través del cerrojo. Éste es otro **ejemplo** de una condición de carrera, en **este** caso una carrera para vencer al reloj. No es posible tolerar una incertidumbre de este tipo en el estado final.

⁷ Sin embargo, éstos encuentran aplicación en sistemas de control y otros circuitos secuenciales que no son síncronos como se explicará en el capítulo 7.

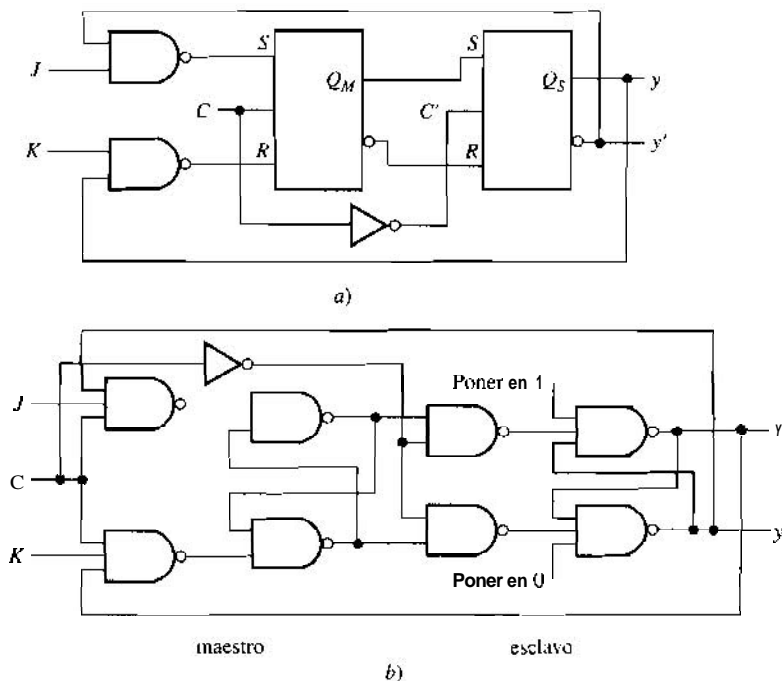


Figura 10. Cerrojo maestro-esclavo.

Un diseño posible

¿Cómo se puede resolver este problema? Lo que necesitamos, una vez que un pulso de reloj ha iniciado una transición, es evitar de alguna manera que la transición se complete hasta que el pulso haya terminado. Existen **varios** diseños que ejecutan esta función; uno de ellos se **presenta** en el diagrama esquemático de la **figura 10d**.

Este circuito recibe el nombre de cerrojo *maestro-esclavo*. La unidad a la derecha (llamada el *esclavo*) tiene una señal de reloj invertida comparada con la de la izquierda (denominada el *maestro*). Esto es, cuando el reloj maestro es 1 el reloj esclavo es 0, y viceversa.

En el diseño real que se indica en la figura 10b, el maestro es similar al diseño JK de la figura 9, salvo en que la retroalimentación en las compuertas de control del maestro se toma desde la salida del esclavo y no desde su propia salida. Estudie con cuidado este diagrama y advierta sus características en comparación con las del cerrojo JK.

Quando la señal del reloj de entrada cambia al nivel bajo, el maestro queda deshabilitado pero el esclavo se habilita (su reloj pasa a alto). Durante este intervalo, la salida del maestro no cambia, aunque la salida del esclavo efectúa una transición hacia cualquiera que haya sido la salida del maestro al principio de este intervalo. Al final del intervalo bajo del reloj, las salidas del maestro y el esclavo se encuentran en el mismo estado, el presente.

Luego, cuando llega el siguiente pulso de reloj, el esclavo se deshabilita y su salida no cambia. La ecuación (3) sigue siendo válida para la transición de estado del maestro. Sin embargo, puesto que no está cambiando la salida del esclavo, el estado presente, retroalimentado al maestro, permanece fijo, sin importar cuanto haya estado el reloj en el nivel alto. Al final del pulso de reloj, se habilita el esclavo y el estado del maestro (el estado siguiente) se transfiere al esclavo. ¡Parece ser que el problema de temporización está resuelto!

En ocasiones es necesario, al principio de una operación (tal como el conteo), fijar los estados iniciales de los cerrojos y flip-flops en el circuito con ayuda de **medios externos**. Un mecanismo para llevarlo a cabo se presenta en la figura 10b en la forma de dos **entradas** directas al

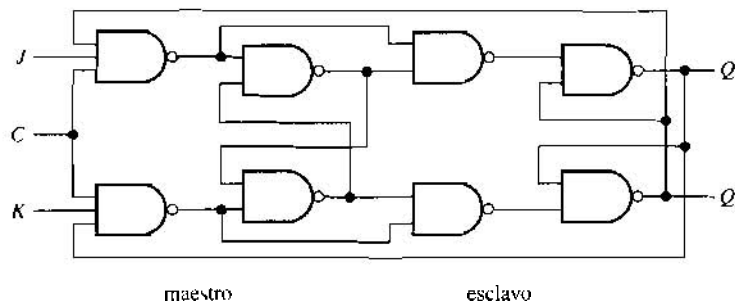


Figura 11. Cerrojo alternativo maestro-esclavo.

esclavo desde el exterior (puesta a 1 y puesta a 0), anteriores a las entradas usuales. Nunca se permite que estas entradas sean 0 de modo simultáneo.⁸

Ejercicio 6. Determine el estado Q_s del esclavo para las combinaciones de entradas de puesta a 1 (PS) y puesta a 0 (B) aparte de 00.

Respuesta⁹

Un diseño maestro-esclavo alternativo

El diseño en la figura 10 no es la única posibilidad; en la figura 11 se incluye otro para el cerrojo JK maestro-esclavo.¹⁰ En este circuito, se eliminan las entradas de reloj invertidas para las compuertas de control del esclavo en el circuito precedente y se sustituyen por las salidas de las compuertas de control del maestro. Éste es el único cambio. Confirme que no ocurre transición de estado en el esclavo cuando $C = 1$ y que $Q_{\text{esclavo}}^+ = Q_{\text{maestro}}$. En consecuencia, el circuito se comporta como un cerrojo maestro-esclavo.

Advierta la secuencia de eventos en el circuito. Los cambios de valores (niveles) lógicos ocurren en las líneas J y K de tiempo en tiempo. En el flanco inicial de un pulso de reloj se inicia una transición de estado. La salida del maestro toma un valor apropiado para el estado presente del esclavo y los valores J y K en este tiempo. Nada ocurre a la salida del esclavo hasta el flanco final del pulso de reloj. Cuando eso sucede, después de un retardo de propagación adecuado, la salida del esclavo adquiere su nuevo valor.

Puesto que el pulso de reloj activa la transición de estado completa (inicio y terminación), decimos que este cerrojo se activa por pulsos.

Desafortunadamente, aún queda un problema con el cerrojo JK maestro-esclavo activado por pulso. Aun cuando la salida del esclavo no puede cambiar en la presencia del pulso de reloj, sí debe ocurrir un cambio en la entrada J o K durante este tiempo; la salida del maestro experimentará otra transición. Entonces, después del flanco final del pulso de reloj, este valor será enviado al esclavo. Por tanto, la transición efectuada por el cerrojo quizá no sea la efectuada por el maestro luego del arriba del pulso de reloj — como debe ser — sino una transición subsecuente. Esta situación tampoco puede tolerarse.

Una solución posible consiste en reducir lo más posible el ciclo de trabajo de la señal de reloj (el ancho del pulso de reloj para una frecuencia determinada). La viabilidad de un cambio en

⁸ En algunos casos, la habilitación y la deshabilitación se llevan a cabo al ir hacia el valor negativo las señales de restablecimiento y borrado. En tales casos estas señales externas entrantes deben invertirse primero. Esto se indica mediante una burbuja a la entrada.

⁹ $Q_s = 1$ para PS = 0, CL = 1; $Q_s = 0$ para PS = 1, CL = 0; operación normal para PS = 1, CL = 1. •

¹⁰ Los diseños que se presentan aquí se incluyen por razones pedagógicas; a menudo se producen consideraciones prácticas en diseños reales diferentes que se disponen comercialmente.

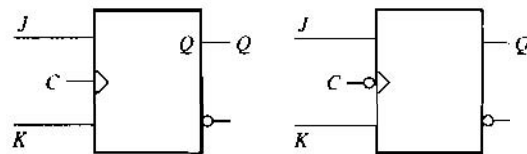


Figura 12. Símbolo esquemático de un flip-flop activado por flanco.

J o K durante este corto tiempo se reducirá de manera correspondiente. En cualquier caso, para evitar la incertidumbre en la salida, los valores de entrada J y K deben mantenerse estables durante el intervalo completo del pulso de reloj.

Parámetros de activación por pulso

Una solución a los problemas anteriores de temporización y carrera es abandonar la activación por pulso en conjunto y diseñar elementos de memoria que se activen sólo mediante un flanco del pulso de reloj. Se dice que estos elementos serán **activados por flanco** y reciben el nombre de **flip-flop**. En un flip-flop activado por flanco, una vez que se carga la información necesaria para la transición de estado (los valores presentes de J, K y el estado), durante un intervalo de tiempo en torno a un flanco (ya sea inicial u final), no tendrán efecto cualesquiera cambios adicionales; afirmamos que estos efectos han sido bloqueados. En otras palabras, un flip-flop no exhibe transparencia en cualquier estado de reloj: las salidas del flip-flop responden al estado de las entradas en un flanco de reloj.

En la figura 2h se muestra una forma realista para un pulso de reloj. (Observela de nuevo.) Se muestran dos intervalos de tiempo en el flanco inicial: El *tiempo de establecimiento* y el *tiempo de retención*. (Ocurren evidentemente intervalos similares en el flanco final.)

El tiempo de establecimiento se extiende desde algún instante anterior a la iniciación del pulso de reloj hasta el inicio de este mismo. El tiempo de retención se extiende desde el final del pulso del reloj hasta algún tiempo posterior luego de que la caída de éste se ha completado. Para la operación adecuada del flip-flop activado por flanco, los valores de las entradas J y K deben permanecer estables desde el inicio del tiempo de establecimiento hasta el final del tiempo de retención. Puesto que este intervalo es mucho más corto que el propio ancho del pulso, resulta mucho más fácil asegurar que no ocurrirá ningún cambio en J o K durante este intervalo.

La activación por flanco se indica en un diagrama esquemático mediante un pequeño triángulo, denominado *indicador dinámico*, ubicado en la terminal del reloj, como se ilustra en la Figura 12. Si la activación ocurre en el flanco inicial, el símbolo es el que se muestra en la figura 12a; la burbuja en la figura 12b indica que la activación ocurre en el flanco final. Aunque no se dará aquí ningún diagrama de circuito para un flip-flop JK activado por flanco, existen este tipo de circuitos (su existencia se supone en lo que sigue y en los problemas.) En lugar de eso, explicaremos un diseño activado por flanco para el tipo de flip-flop que se describirá a continuación."

Flip-flops de retardo (D)

La condición que se requiere en (2) para la operación adecuada de un cerrojo SR, que $S \cdot R = 0$ siempre, puede conseguirse de manera automática si la entrada R se obtiene de la entrada S mediante un inversor de manera que $R = S'$. El resultado es un circuito con una entrada externa simple en la cual S se marca ahora con la letra D (representando *dato*). Una versión simple se muestra en la figura 13a. El pulso de reloj actúa como una compuerta que, cuando está en el estado alto, permite que los datos en D pasen hacia la salida. Visto de este modo, este circuito

¹¹ Un flip-flop JK característico activado por flanco es el SN74111. Éste tiene un tiempo de establecimiento de 0 y un tiempo de retención de 30 ns. Los tiempos totales de establecimiento y retención para los otros flip-flops en la tecnología TTL corresponden también a alrededor de 30 ns.

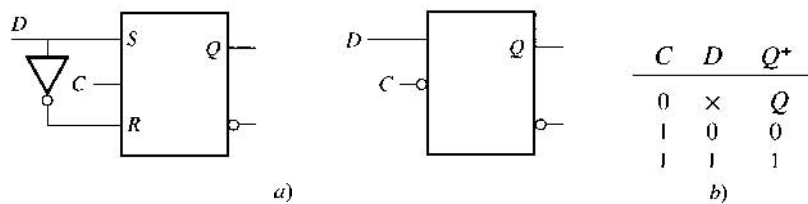


Figura 13. Cerrojo D.

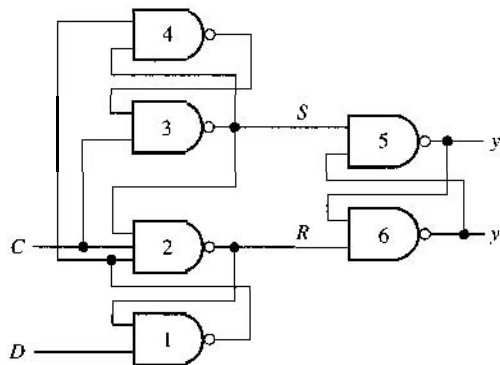


Figura 14. Diseño del flip-flop D activado por flanco.

algunas veces se denomina **cerrojo de compuerta**. ("Cerrojo" debido a que el reloj se considera como una compuerta que deja pasar los datos a través de ella cuando se habilita.)

La ecuación de transición para el cerrojo D puede obtenerse de la correspondiente al cerrojo SR dejando a $R = S'$ y luego renombrando S como D en (2). El resultado es

$$Q(t+1) = D(t) \quad (5)$$

Lo que nos indica esta expresión es que el estado siguiente será cualquiera que corresponda a la entrada D durante el intervalo del pulso de reloj. (En consecuencia, D también puede indicar *re-tardo*.) O, mirando hacia atrás en el tiempo a partir del presente, cualquiera que sea el estado del cerrojo D en el tiempo del pulso de reloj presente, éste es el que correspondía a su entrada durante la aparición del pulso de reloj previo. Advierta que, mientras que cada uno de los cerrojos SR y JK tienen dos entradas (excitaciones), sin contar el reloj, el cerrojo D sólo tiene una.

Ejercicio 7. Utilizando la ecuación de transición para el cerrojo SR, confirme la tabla de transición dada en la Figura 13b. ♦

Aunque el cerrojo D se crea a partir del SR en la figura 13, es posible obtener el mismo comportamiento si el cerrojo SR se sustituye por uno JK. Esto es, **se fuerza** a que K sea igual a J' mediante un inversor. (Compruebe que $Q^+ = D$ si J se renombra como U.) El cerrojo tipo D tiene los mismos problemas de temporización que el cerrojo JK. Sin embargo, es un dispositivo simple y económico que encuentra aplicaciones cuando no es necesario sincronizar todas las transiciones de cerrojo en un sistema.

Flip-flop D activado por flanco

La solución que se mencionó para el problema de temporización de cerrojos activados por pulso es la **activación por flanco**. Esto es, en un flip-flop D, los cambios en el valor de entrada D no afectarán el estado salvo en el flanco del pulso de reloj, ya sea éste el ascendente o el descendente, dependiendo del diseño. En la figura 14 se presenta el diseño de un flip-flop D activado por flanco. Las compuertas 5 y 6 constituyen un cerrojo SR básico.

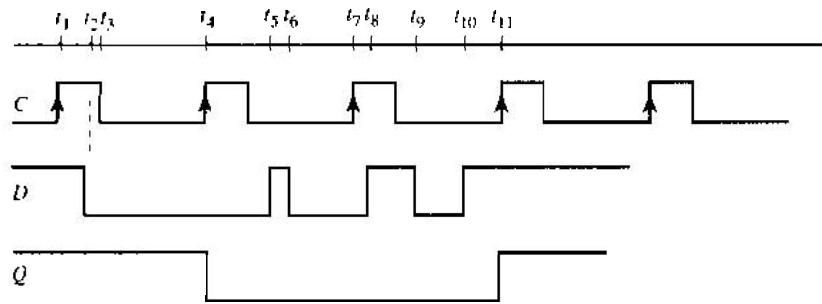


Figura 15. Formas de onda del flip-flop D activado por flanco.

Las expresiones para las entradas S y R para este cerrojo pueden obtenerse analizando el resto del circuito, consistente en las compuertas de la 1 a la 4. Analice el circuito para confirmar lo siguiente:

$$\begin{aligned} S^+ &= C(S + DR') \\ R^+ &= CS'(D' + R) \end{aligned} \quad (6)$$

Cuando la señal de reloj está en el estado bajo ($C = 0$), estas expresiones muestran que tanto S como R^+ son 0, independientemente de la entrada D . De acuerdo con (2), $Q^+ = Q$, no ocurren transiciones, e y mantiene cualquiera que haya sido el valor que tenía: $y^+ = y$.

Suponga ahora que el reloj asciende a 1: de acuerdo con (6) y debido a que S y R fueron ambas 0 justo antes de ese hecho, $S^+ = D$ y $R^+ = D'$. Después de que ocurre un retardo necesario para esta transición, el uso de estos valores para S y R en (2) produce $Q^+ = D + DQ = D$.

Resumiendo, en el flanco ascendente del pulso de reloj (después de un retardo apropiado), la salida del circuito se convierte en cualquiera que haya sido la entrada D en el momento del flanco ascendente.¹²

Suponga ahora que finaliza el pulso de reloj: $C = 0$. Ya vimos que los valores de S y R se volverán 0 pero que no ocurrirá ninguna transición en la salida: $Q^+ = Q$. No ocurrirá otro cambio adicional en la salida hasta el flanco ascendente del siguiente pulso de reloj. La salida subsiguiente (después de un retardo) será cualquiera que sea el valor de entrada en ese tiempo.

En la figura 15 se muestran formas de onda posibles para la señal de entrada (D) y el reloj (C). También se indica la forma de onda de salida que resulta, bajo la suposición de que el flip-flop estaba inicialmente establecido ($Q = 1$). La no idealización del pulso de reloj se omite en esta figura. (Tenga presente la situación atípica.) Tanto D como Q son 1 en el primer flanco ascendente del reloj (t_1), por lo que, ya que $Q^+ = D = 1$, no ocurre transición. (Compruebe cada punto en éste y en el siguiente párrafo.)

En el siguiente flanco ascendente (t_4), $Q^+ = D = 0$, y se produce una transición. En el siguiente flanco ascendente (t_7), D es otra vez 0, por lo que no ocurrirá transición en la salida. Cuando se aproxima t_{10} , $Q = 0$ y $D = 1$, de manera que una transición hacia 1 ocurre de nuevo en el flanco ascendente del pulso de reloj. En todos los tiempos precedentes cuando cambia el valor de D (t_5 , t_6 , t_8 , y t_{10}), el reloj está estable, por lo que no se producen transiciones en la salida.

Como comentario final, ¿qué pasaría si fuera a cambiar la entrada D durante el intervalo sobre el cual sube el pulso de reloj? La respuesta es, ¡el caos! Esto se debe a que el "flanco" tiene una pendiente finita. No sería del todo claro qué valor tendría la entrada durante este "flanco" lento, por lo que la salida sería incierta. Ésta es la razón del requerimiento de que la entrada permanezca estable (sin cambio) durante un intervalo de tiempo que abarque los tiempos de esta-

¹² Funcionalmente, "el valor de D en el flanco ascendente" significa que D debe mantener su valor durante un tiempo no menor que el tiempo de establecimiento más el tiempo de retención alrededor del flanco de ascenso.

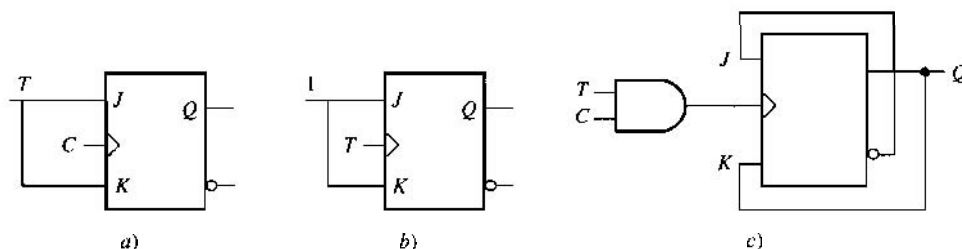


Figura 16. Diferentes diseños para un flip-flop biestable "T".

blecimiento y retención del flip-flop. En la figura 15, por ejemplo, no se puede permitir el cambio en D de 0 a 1 cerca de t_7 si el aumento en el pulso de reloj no ha terminado.

Flip-flop T

Un dispositivo que sería útil en sistemas digitales es aquél cuya salida se alterne. Esto es, cuya salida se sustituya por su complemento siempre que haya una señal entrante. Un dispositivo de este tipo sería inherentemente de entrada simple. El dispositivo que presenta este comportamiento se conoce como *flip-flop biestable (T)*.

No hay necesidad de un diseño independiente debido a que un flip-flop de este tipo se obtiene con facilidad a partir del flip-flop JK con ciertas conexiones en las terminales.

En la figura 16a se muestra un diseño posible. Las entradas J y K de un flip-flop JK se conectan entre sí y se renombran T . La tabla de transición se obtiene sin dificultad a partir de la correspondiente al flip-flop JK en la figura 9. Para $C = 1$, la ecuación de transición en (4) se reduce a:

$$Q^* = TQ' + T'Q = T \oplus Q \quad (8)$$

Para $T = 1$ (cuando $C = 1$), $Q^* = Q'$; esto es, el nuevo estado es el complemento del estado anterior. El estado se alterna. Pero si $T = 0$ cuando $C = 1$, entonces $Q^* = Q$; no hay alternancia. Puesto que, en la presencia de un pulso de reloj, la alternancia ocurre sólo cuando $T = 1$ y no cuando $T = 0$, este circuito es un poco deficiente.

Ejercicio 8. Otro diseño para un flip-flop T síncrono se ilustra en la figura 16c. La señal J se retroalimenta desde Q' y K se retroalimenta desde Q . (Recuerde que éste es un diagrama de bloques; en el circuito de flip-flop JK real de la figura 9, esto equivale a conectar las terminales J y K a un 1 lógico.) Determine la ecuación de transición para este circuito y compárelo con el de la figura 16a tanto para $C = 0$ como para $C = 1$.

Respuesta¹³

Una modificación del diseño de la figura 16a que supera la deficiencia se muestra en la figura 16b. Las terminales J y K se conectan de nuevo, pero ahora se dejan permanentemente en 1. La entrada T se limita en este caso a ser un pulso y se introduce en la terminal del reloj en lugar del reloj. El flip-flop resultante es *asíncrono, sin reloj*. Para determinar la ecuación de transición, se fija $J = K = 1$ y $C = T$ en (4) el resultado es el mismo que (8). También en este caso, la alternancia ocurre cuando $T = 1$, pero ahora éste es para toda ocurrencia del pulso T . Esto es, cada vez que el pulso de entrada T pasa al estado alto, la salida se alterna.

Hemos considerado brevemente dos diseños de flip-flop T. Uno es síncrono pero no se alterna en cada pulso de reloj; el otro se alterna en cada pulso de entrada pero no es síncrono.

¹³ La misma

Requerimientos de excitación del flip-flop

Las partes previas de este capítulo pueden resumirse de la manera siguiente. A partir de un diseño dado de flip-flop, es posible determinar una tabla de transición de estados, o la equivalente ecuación de transición, mediante el análisis del circuito. A partir de cualquiera de éstas, se puede determinar el estado siguiente para cada estado presente y cada combinación de entradas.

En un diseño de circuito secuencial, el cual que explicará en el capítulo 6, no se conocen las excitaciones del flip-flop. En vez de eso, para cada diseño y cada combinación de entradas, lo que se conoce son tanto el estado presente como el estado siguiente. A partir de esta información, debemos deducir los valores de excitación requeridos que se producirán en una transición determinada.

EJEMPLO 1

Suponga que se requiere una transición de $Q = 1$ a $Q^+ = 0$ para un flip-flop JK. Si estos valores se insertan en la ecuación de transición (4), el resultado será $Q^+ = JQ' + K'Q$ o $0 = 1 \cdot J + 1 \cdot K' = K'$. Por lo que $K = 1$, independientemente de J . Esto es, si se desea la transición $Q = 1 \rightarrow Q^+ = 0$, las entradas requeridas del flip-flop JK son $K = 1$, J sea la que sea. ■

Los requerimientos de excitación para las otras transiciones pueden determinarse de manera similar. Los resultados para los flip-flops que se están considerando se ilustran en las tablas de excitación de la figura 17.

Transición requerida		Entradas necesarias	
e	Q^+	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

$Q^+ = S + R'Q$, $S \cdot R = 0$
a) Flip-flop SR

Transición requerida		Entradas necesarias	
e	Q^+	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

$Q^+ = J'Q + K'Q$
b) Flip-flop JK

Transición requerida		Entradas necesarias	
Q	Q^+	D	
0	0	0	
0	1	1	
1	0	0	
1	1	1	

$Q^+ = D$
c) Flip-flop D

Transición requerida		Entradas necesarias	
Q	Q^+	T	
0	0	0	
0	1	1	
1	0	1	
1	1	0	

$Q^+ = T \oplus Q$
d) Flip-flop T

Figura 17. Requerimiento de excitación del flip-flop.

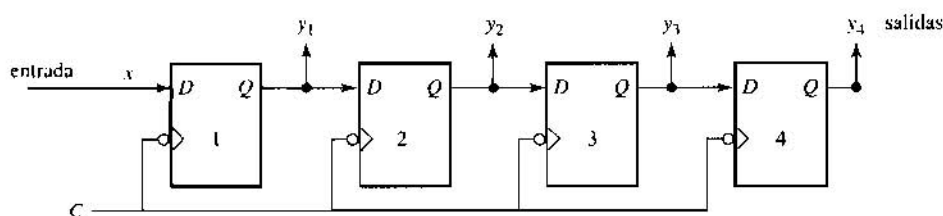


Figura 18. Registro de corrimiento hacia la derecha.

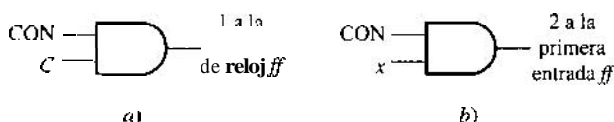


Figura 19. Control de entrada de registro de corrimiento.

Ejercicio 9. Utilizando las ecuaciones de transición apropiadas para cada uno de los flip-flops considerados, confirme cada una de las tablas de requerimientos de excitación de la figura 17. *

3 REGISTROS

Las siguientes características distinguen un circuito secuencial de uno combinatorio:

- La capacidad para almacenar, en memoria, información acerca del estado del circuito debido a entradas pasadas.
- La utilización de esta información para producir una salida en respuesta a nuevas entradas.

Las unidades básicas para el almacenamiento de 1 bit de información son flip-flops o cerrojos. Del mismo modo que las compuertas lógicas pueden interconectarse para constituir unidades más grandes como multiplexores, decodificadores y sumadores completos, así los flip-flops se organizan en grupos llamados registros. Un registro de n bits es un conjunto de n flip-flops (casi siempre del tipo D), todos con un reloj común. No sólo debe ser común la señal de reloj, sino que todos los flip-flops deben responder al reloj de la misma manera: todos activados por el flanco ascendente o por el flanco descendente.

Es posible almacenar n bits de información que pueda o no relacionarse. La transferencia de información a un registro se conoce como carga del registro. En términos conceptuales, es posible transferir la información a todos los flip-flops en un registro de manera simultánea (en paralelo) o un bit a la vez (en serie). De modo similar, la información puede transferirse hacia fuera del registro en paralelo o en forma serial. Resultan viables las cuatro combinaciones de carga y lectura: Paralelo-entrada/paralelo-salida, Paralelo-entrada/serie-salida (liste usted las demás).

Los registros se pueden obtener en circuitos MSI. La información se procesa en unidades de 2ⁿ bits. (Cuanto más alto este número, tanto más rápido el procesamiento.) Con fines ilustrativos y facilidad de visualización en lo que sigue, se utilizarán registros con menos de 8 (2^3) flip-flops.

Registro de corrimiento de carga en serie

El diagrama esquemático para un registro en serie de 4 bits se muestra en la figura 18. Los flip-flops indicados son del tipo D, aunque también podrían ser JK. El indicador dinámico (triángulo) indica un flip-flop activado por flanco, y la burbuja sobre la terminal de entrada de reloj

quiere decir que la activación ocurre en el flanco descendente del pulso de reloj. Vamos a hacer varias observaciones luego de examinar el diagrama.

En cada flanco descendente del pulso de reloj, la entrada en la línea x se transfiere a la salida del primer flip-flop. Cualquiera que sea la salida del primer flip-flop en ese tiempo se transfiere a la salida del segundo flip-flop, y de manera similar, en una cadena hacia la derecha, hasta el último flip-flop. ¿Funcionaría este esquema si hubiera habido más flip-flops en esta cadena? Esto es, los datos se corren con cada pulso de reloj, de ahí el nombre *registro de corrimiento*. Si la única salida disponible externamente es la última a la derecha (o a la izquierda en el caso de corrimiento izquierdo), entonces el registro recibe el nombre de *registro de corrimiento de salida en serie*. Sin embargo, si cada salida de flip-flop (cada Q , no sólo Q_4) está disponible para leerse externamente, este es también un *registro de salida en paralelo*. A menudo están disponibles ambas posibilidades, y se usan señales de control específicas para controlar cuál de estas formas se utilizará en una aplicación particular.

En virtud de que la transferencia de datos ocurre con cada pulso de reloj, en el circuito de la figura 18 no hay control sobre la temporización de la transferencia de datos. Dicho control se puede conseguir en una de dos formas. Una posibilidad se ilustra en la figura 19a, donde CON es la señal de entrada del control de corrimiento. Debido a que CON se introduce a una compuerta AND con el reloj, los flip-flops en el registro de corrimiento se habilitan sólo cuando $CON = 1$. En un registro de k bits, CON debe pasar al estado alto justo después de un flanco posterior del pulso de reloj y permanecer en el nivel alto durante k periodos de reloj. El registro operará como se describió antes sólo en estos k periodos de reloj.

Por tanto, una palabra de k bits se transferirá al registro. La señal de control de corrimiento cambia para $CON = 0$ en el extremo de la palabra de k bits, deshabilitando de ese modo los flip-flops en el registro hasta que la señal de control vuelva de nuevo al estado alto.

La principal desventaja de este esquema es que la lógica se efectúa con la señal de reloj. Puesto que las compuertas con las cuales se efectúa la lógica (una compuerta AND en este caso) tienen retardo de propagación, la señal de reloj no llegará a todos los flip-flops en el sistema al mismo tiempo. Por consiguiente, el sistema quizá falle al efectuar todas las funciones del sistema en forma síncrona —un resultado altamente indeseable.

Una manera simple de superar esta dificultad de control de la entrada se muestra en la figura 19b. En vez de hacer pasar la señal de control por una compuerta AND con el reloj, la señal se hace pasar por una AND con la entrada x . También en este caso, CON debe pasar al estado alto justo después de un flanco posterior del pulso de reloj, permaneciendo en ese mismo estado alto durante k periodos de reloj, y luego pasar al estado bajo. De esta manera, sólo se transferirá una palabra de k bits al registro de corrimiento. Este método para el control de carga se ilustra a continuación.

Registro de corrimiento de carga en paralelo

El diagrama esquemático de un registro de entrada en paralelo, salida en paralelo, se presenta en la figura 20. Se diseña con flip-flops JK de activación por flanco y utiliza el método de control de carga que se explicó antes. Observaremos algunas de sus características estudiando el diagrama.

1. El pulso de reloj se aplica a cada uno de los flip-flops a través de un inversor, de manera tal que la activación se efectúa sobre el flanco final del pulso de reloj. El propósito principal del inversor, sin embargo, es proporcionar un reforzamiento de la señal, reduciendo de ese modo la carga sobre la fuente de reloj: el reloj no tendrá que accionar todos los flip-flops, únicamente al inversor.
2. El esquema de control de entrada en la figura 20 se usa con cada señal de control. Esta vez, para evitar la carga de la fuente de control, la señal se introduce a través de un búfer.

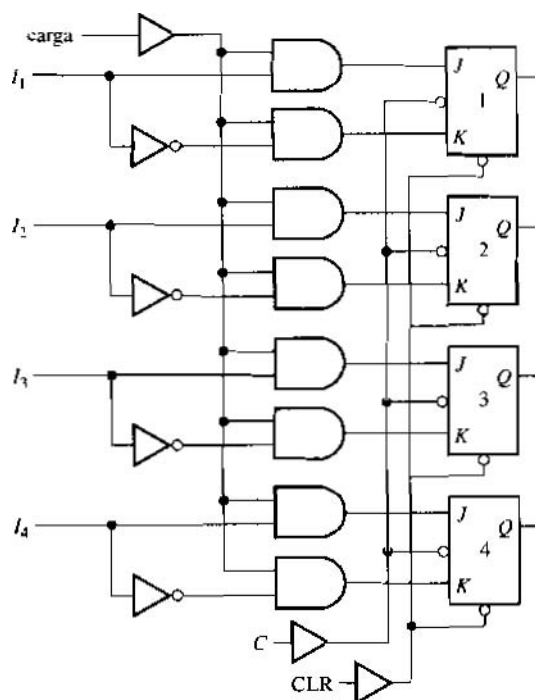


Figura 20. Registro de 4 bits cargado en paralelo con control de carga.

3. Por último, existe una entrada de BORRAR (CLR) operada de manera independiente que se aplica a cada flip-flop en el registro; ésta debe estar en alto en la operación normal del registro. Sin embargo, cuando es necesario anular la función que se está efectuando y borrar los contenidos del registro, ello se logra, en forma asíncrona, fijando $CLR = 0$. (¿Cuál es la función del búfer en esta línea?)

Aunque los flip-flops son JK , cuando en la entrada de control de carga es $L = 1$, éstos se están utilizando como flip-flop D debido a que en ese caso $K = J'$. (Compruébelo.) Cuando $L = 0$, sin embargo, $J = K = 0$; en consecuencia, no ocurrirán transiciones de flip-flop. Las entradas I_i se cargan de manera simultánea en los flip-flops correspondientes en el flanco final del pulso de reloj, siempre y cuando la entrada CLR sea 1 en ese tiempo y también sea 1 el control de carga. Todas las salidas del flip-flop están disponibles externamente, de manera que éste es un registro de entrada en paralelo, salida en paralelo. Por simplicidad, los registros que se muestran aquí son de 4 bits.

Evidentemente, es posible realizarlos para más bits agregando un número mayor de flip-flops. El número de flip-flops es la única diferencia entre los registros que procesan $16(2^4)$ bits $-32(2^5)$ o 2^n para un valor superior de n - y los más simples que se exponen aquí.

La totalidad de este tipo de registros se obtienen en CI MSI. La figura 21 muestra un símbolo esquemático para un registro de entrada paralelo, salida paralelo, de 8 bits utilizando flip-flops D y entrada CLR.

Conversión paralelo-serie

Aunque todos los bits de una palabra estén quizá disponibles al mismo tiempo, tal vez resulte deseable convertir esta información en la forma serie. Ese es el caso, por ejemplo, si los datos se van a transmitir serialmente a otra localidad por un canal de comunicación de una sola línea.

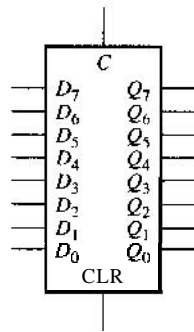


Figura 21. Símbolo para un registro de 8 bits.

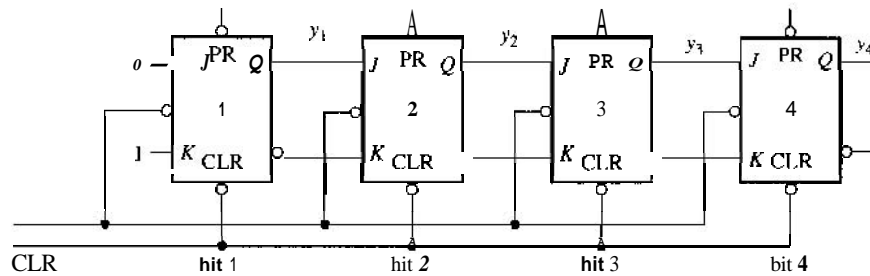


Figura 22. Registro de corrimiento de 4 bits para la conversión paralelo-serie.

Pulso de reloj	y_1	y_2	y_3	salida de serie y_4
	(palabra de entrada)			(1sb de entrada)
1	1	1	0	1
2	0	1	1	0
3	0	0	1	1
4	0	0	0	1
	palabra de salida			(1sb de salida)

Figura 23. Conversión de datos paralelo-serie.

Con este fin, se necesita que un registro de corrimiento pueda cargarse en paralelo. Una posibilidad se presenta en la figura 22. El registro de corrimiento de 4 bits está conformado por flip-flops JK , aunque éstos actúan como flip-flops D , pues $K_i = J_i$. Cada flip-flop tiene también terminales asíncronos BORRAR O PONER A 0 (CLR) y PRESTABLECER O PONER A 1 (PR).

Los datos entran al registro a través de las terminales PRESTABLECER 1. (No se indican las unidades de control de carga en cada entrada PRESTABLECER A 1.) Las entradas J y K al primer flip-flop a la izquierda se fijan de manera permanente en los estados bajo (0) y alto (1), respectivamente. Esto garantiza que después de cada pulso de reloj, la salida del primer flip-flop pasará al estado bajo. A menos que los estados del flip-flop se fijen asincrónicamente, este valor bajo se propagará hacia la derecha y, luego de tres pulsos de reloj más, todos los flip-flops se borrarán. En un tiempo determinado por la entrada de control de carga, el registro se carga a través de las terminales PRESTABLECER A 1.

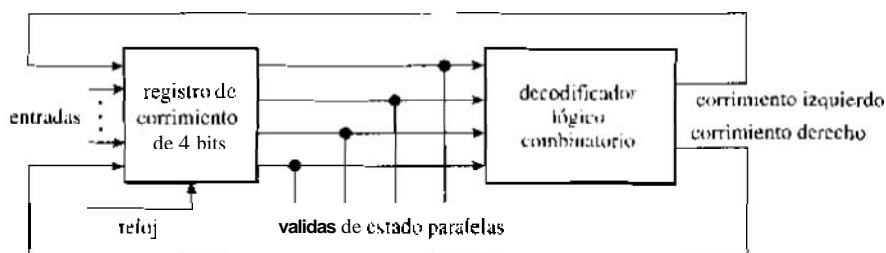


Figura 24. Registro universal

Suponga que la palabra **que** se va a transmitir es 1101. En un pulso de reloj determinado, la entrada de **control de carga** habilita las terminales PRESTABLECER A 1, y todos los bits de la palabra se cargan de manera simultánea en el registro. El resultado se indica en el primer renglón de la tabla en la figura 23. PRESTABLECER A 1 está ahora deshabilitada. En cada pulso de reloj sucesivo, los contenidos del registro se corren hacia la derecha, dejando 0 a la izquierda, ~ o ~ como se muestra en los renglones sucesivos de la tabla. La salida, **que** aparece en la última columna, se toma desde el flip-flop más a la derecha, el primer bit menos significativo.

Otra **aplicación** de este tipo de registro de conversión paralelo-serie es la siguiente: suponga que se disponen dos números binarios (que se **sumará**) en forma paralela, pero que se procesarán mediante un sumador en serie. Los números pueden convertirse en la forma serial y aplicarse como entradas seriales en el sumador.

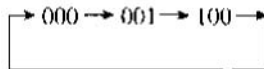
Registros universales

Cada tipo de registro descrito en las secciones anteriores (corrimiento a la derecha, corrimiento a la izquierda, carga en paralelo, lectura en paralelo) tiene aplicaciones importantes. El que sería de un valor incluso mayor es un registro que combina **algunas** de todas estas características, un tipo de registro universal.

En la figura 24 se presenta el diagrama esquemático para un registro de 4 bits de este tipo. Hay cuatro terminales para la carga en paralelo y la lectura en paralelo, así como entradas en serie de corrimiento izquierdo y corrimiento derecho. El propósito del decodificador externo es producir los bits que se correrán y controlar la dirección en la cual se efectuará el corrimiento.

Una de las aplicaciones de este tipo de circuito es la que corresponde a un contador. Puesto que los bits 0 y 1 pueden correrse hacia cualquier dirección en el pulso de reloj, es posible generar un gran número de códigos diferentes de longitud variable. Los problemas para llevar a cabo tales diseños se exponen en el capítulo 6.

Para ilustrar, suponga que se va a utilizar un registro universal de 3 bits para diseñar un contador de módulo 3 de acuerdo con el siguiente código. (¿En éste un código de distancia unitaria?)



Suponga que las variables de estado se denominan y_1, y_2, y_3 , con y_1 representando el bit más significativo. Empezando con el estado $y_1 y_2 y_3 = 000$, un bit 1 se corre hacia la izquierda, luego un bit 1 hacia la derecha, después un bit 0 hacia la izquierda. Los mapas lógicos para los datos que se van a correr a la izquierda y a la derecha se construyen en la forma que se indica en la figura 25a.

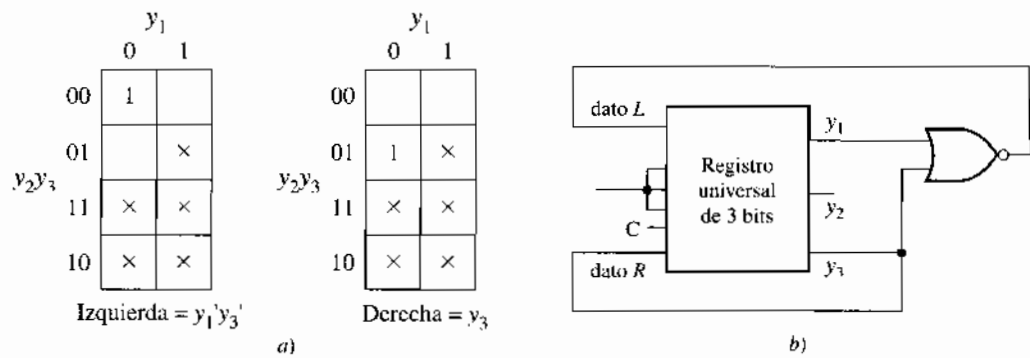


Figura 25. Contador de modulo 3 utilizando un registro universal.

Como ejemplo, en el estado presente 000 (celda 000 en el mapa), el bit que se va a correr hacia la izquierda es un 1, y el que se correrá hacia la derecha, un 0. Puesto que el corrimiento se especifica para sólo tres valores presentes de las variables de estado, las entradas en cinco de las celdas del mapa son valores no relevantes. (Confirme todas las entradas en los mapas.) Se obtienen con facilidad expresiones para las funciones de corrimiento hacia la izquierda y corrimiento hacia la derecha; compruebe las que se dan según los mapas. El contador completo se muestra en la figura 25b. Las terminales de entrada en paralelo están inactivas y la totalidad de las mismas se conectará al voltaje bajo o alto del sistema dependiendo de la construcción interna del registro.

RESUMEN Y REPASO DEL CAPÍTULO

Este capítulo presentó los circuitos secuenciales. Aborda los dispositivos de almacenamiento de memoria: flip-flops y registros. Se cubrieron los siguientes temas.

- Problemas de temporización y relojes.
- Operación del cerrojo *SR* básico.
- Retroalimentación.
- El concepto de estado del circuito y su cambio al estado siguiente.
- Ecuaciones de transición de estado y tablas de transición.
- Cerrojos *SR* sin reloj y con reloj, y el problema de entradas simultáneas en estado alto.
- El cerrojo *JK* y su ecuación de transición.
- El problema de carrera y procedimientos para resolverlo.
- Diseño maestro-esclavo y activación por flanco.
- Flip-flops *D*.
- Flip-flops *T*.
- Requerimientos de excitación para cada tipo de flip-flop.
- Registros de corrimiento.
- Carga de registros en serie.
- Carga de registros en paralelo.
- Conversión de carga de paralelo a serie y viceversa.
- Registros universales.

PROBLEMAS

Usted será capaz de efectuar parcialmente estos problemas sólo después de estudiar las últimas secciones del capítulo. Conserve aquellas partes de sus soluciones que realice primero; después complete los problemas luego de estudiar el material final.

1. Se va a definir un cerrojo con entradas L y M (un cerrojo LM). En la figura P1a se proporciona una tabla que especifica el estado siguiente deseado del pulso de reloj.

L	M	Q	Q^+	N	P	Q	Q^+	G	H	Q	Q^+
0	0	0	0	0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	1	1	0	0	1
1	0	1	1	1	0	1	1	1	0	1	0
1	1	0	1	1	1	0	0	1	1	0	0
1	1	1	0	1	1	1	1	1	1	1	0
a)				b)				c)			

Figura P1 Tablas de transición para tres cerrojos. a) Cerrojo LM . b) Cerrojo NP . c) Cerrojo GH .

- Describa cómo cambiará el estado del cerrojo en cada combinación de valores de L y M . a) Por ejemplo, para $LM = 10$, el estado siguiente es 1, independientemente del estado presente.)
 - Escriba expresiones para el estado siguiente Q^+ en términos de L , M y el estado presente Q .
 - Construya la tabla de requerimientos de excitación.
 - Elabore el diagrama de un circuito que cumple con la tabla de transición correspondiente utilizando un cerrojo SR con reloj y cualesquiera compuertas adicionales que sean necesarias.
 - Repita la parte d utilizando un cerrojo JK .
- Repita el problema 1 para el cerrojo NP definido mediante la tabla de transición de la figura P1b.
 - Repita el problema 1 para el cerrojo GH definido mediante la tabla de transición de la figura P1c.
 - Un cerrojo AB se construye a partir de un cerrojo SR como se indica en la figura P4.

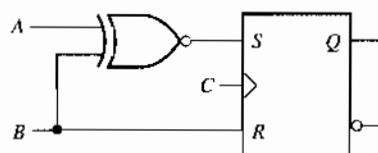


Figura P4

- Obtenga expresiones para S y R en términos de A y B .
 - Escriba una expresión para el estado siguiente Q^+ en términos de A y B y el estado presente Q .
 - Construya la tabla de requerimientos de excitación para A y B .
- El diagrama esquemático de un flip-flop G se muestra en la figura P5. Las transiciones permitidas en este flip-flop son:

$$Q^+ = Q \text{ cuando } G = 0, \quad Q^+ = 1 \text{ cuando } G = 1$$

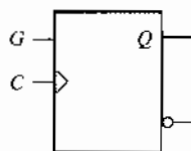


Figura P5

- a. Escriba una expresión para el estado siguiente Q^+ en términos de G y el estado presente Q .
 b. Construya la tabla de requerimientos de excitación para G .
6. El diagrama esquemático de un flip-flop H se muestra en la figura P6. Las transiciones permitidas son:
- $$Q^+ = 0 \text{ cuando } H = 0, \quad Q^+ = Q' \text{ cuando } H = 1$$

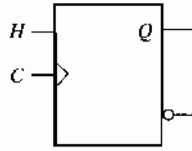


Figura P6

- a. Escriba una expresión para el estado siguiente Q^+ en términos de H y el estado presente Q .
 b. Construya la tabla de requerimientos de excitación para H .
7. Un cerrojo tipo D puede construirse a partir de un cerrojo SR y un inversor forzando $R = S'$. La figura P7 muestra otro tipo de cerrojo con compuerta.

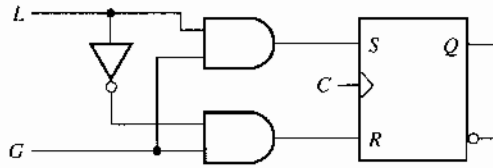


Figura P7

- a. Encuentre una expresión para Q^+ en términos de G y L .
 b. A partir de esto, determine expresiones para Q^+ para $G = 0$ y $G = 1$.
 c. Describa cómo G controla la transferencia de información de L a la salida del cerrojo.
8. En el cerrojo SR con reloj, la condición simultánea $S = 1, R = 1$ produce un estado incierto. Una variación del cerrojo SR es el cerrojo *dominado por la entrada S (SD)*. Para este cerrojo, las entradas simultáneamente altas dan lugar al ajuste del cerrojo, esto es, fijando $Q = 1$.
- a. Construya una tabla de transición para el cerrojo SD . A partir de esto, determine una expresión para el estado siguiente.
 b. Construya un diagrama lógico que ponga en práctica este cerrojo.
9. El circuito de la figura P9 se ha propuesto como un generador de reloj. Se supone que se inserta un pulso mediante algún mecanismo externo en la entrada de uno de los tres inversores más a la izquierda, esto es, en el punto A, B o C .

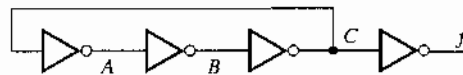


Figura P9

- a. Suponga que los inversores son ideales, con retardo 0. Describa la anomalía resultante.
 b. Suponga ahora que cada inversor tiene un retardo de 25 ns. Utilice diagramas de temporización para analizar el circuito y determinar la salida f . Especifique la frecuencia de este reloj.
10. Analice el flip-flop D activado por flanco que se muestra en la figura 14 del texto, procediendo de acuerdo con los pasos señalados más adelante. Suponga que el retardo de la compuerta corresponde a 1% del periodo del reloj y elabore un diagrama de temporización a medida que avance su análisis.
- a. Con el reloj en el valor bajo encuentre los valores lógicos de todas las variables intermedias con $D = 0$. Verifique que el flip-flop será estable en cualquier estado.
 b. Suponga ahora que la entrada D cambia de 0 a 1 con el reloj aún en el valor bajo. De nuevo determine el valor de las variables intermedias. ¿Existe algún cambio en el estado del flip-flop?

- c. A continuación considere que el reloj pasa de 0 a 1 mientras que $D = 0$. Determine los cambios en las variables intermedias y en la salida del flip-flop. (Podría resultar útil utilizar el retardo de compuerta como una unidad de tiempo a fin de seguir los cambios en las variables intermedias y en la salida del flip-flop.)
- d. Repita la parte c, esta vez con $D = 1$.
11. Un contador de rizo se conforma a partir de flip-flops (T) biestables en los cuales la entrada será la terminal de reloj, mientras que las terminales J y K se mantienen en 1 lógico, como en la figura 16b en el texto. En la figura P11 se presenta el diagrama esquemático de un contador de rizo de 3 bits. Puesto que no hay reloj del sistema, éste no es un contador síncrono. Suponga que cambian las salidas del flip-flop, con un retardo de d_f , en el flanco ascendente de los pulsos en la terminal del reloj.

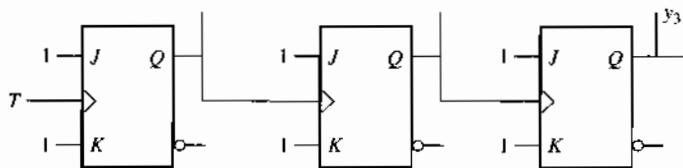


Figura P11

- a. Describa la operación de este contador de rizo. En particular, suponga que el conteo lo va a utilizar algún otro circuito. ¿Será siempre correcto el conteo que muestre dicho circuito?
- b. ¿Cuál es la frecuencia de conteo máxima del contador de 3 bits?
- c. En un contador de rizo de 10 bits suponga que $d_f = 40$ ns. Analice la velocidad máxima a la cual puede cambiar la señal de entrada (T).
12. Suponga que se cuenta con un flip-flop JK . Dibuje un diagrama que incluya un flip-flop JK y una o más compuertas lógicas por construir:
- Un flip-flop D
 - Un flip-flop T
13. Suponga que se cuenta con un flip-flop D . Dibuje un diagrama que incluya un flip-flop D y una o más compuertas lógicas por construir:
- Un flip-flop T
 - Un flip-flop JK
14. Suponga que se cuenta con un flip-flop T . Dibuje un diagrama que incluya un flip-flop T y una o más compuertas lógicas por construir:
- Un flip-flop D
 - Un flip-flop J
15. Un circuito secuencial se integra a partir de tres flip-flops biestables con entradas denominadas T_0 , T_1 y T_2 , y salidas Q_0 , Q_1 y Q_2 , respectivamente. Cada uno de éstos se diseña utilizando un flip-flop JK cuyas entradas J y K se fijan ambas en 1, dejando la terminal del reloj como la única entrada externa. La entrada al flip-flop 1 a la izquierda es una señal de reloj C . La entrada al flip-flop a su derecha proviene de la salida de una compuerta AND de dos entradas, siendo estas últimas C y Q_0 . La entrada al último flip-flop viene de la salida de una compuerta AND de tres entradas, las cuales corresponden a C , Q_0 y Q_1 . Las salidas del flip-flop constituyen una palabra $Q_2Q_1Q_0$. Suponga que todas las salidas de flip-flop son 0 inicialmente.
- Dibuje el diagrama de circuito apropiado utilizando compuertas AND y flip-flops T .
 - Escriba expresiones para cada entrada de flip-flop.
 - Utilizando flip-flops biestables, determine los valores del estado siguiente de las salidas de flip-flop luego de la ocurrencia del primer pulso de reloj, y escriba la palabra de salida.
 - Empezando a partir del estado 000, repita esto para diez pulsos de reloj sucesivos y liste las palabras de salida una luego de la otra. Describa lo que sucede después de ocho pulsos de reloj.

- e. Suponga que las palabras de salida son palabras de código. Examinando las salidas consecutivas, especifique de qué código se trata. ¿Qué operación es la que parece efectuar este circuito?
- f. Suponga que se agrega un cuarto flip-flop biestable a la derecha, siendo la entrada a este mismo la salida de una compuerta AND de cuatro entradas, siendo éstas C , Q_0 , Q_1 y Q_2 . Dibuje el circuito y especule acerca de su naturaleza. Repita la parte d para 20 pulsos de reloj sucesivos y verifique su especulación.

16. El circuito en la figura P16 efectúa cierta función. El objetivo de este problema consiste en determinar cuál es dicha función.

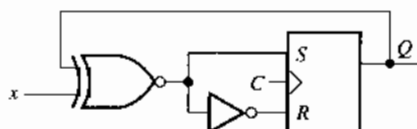


Figura P16

- a. Determine una expresión para el estado siguiente Q^+ en términos de la entrada x y el estado presente Q .
- b. Deje que los pulsos de reloj se numeren 1, 2, 3,... después del punto en el que el flip-flop se pone en cero ($Q_0 = 0$), y deje que x , Q y Q^+ tomen estos subíndices para designar el pulso de reloj durante el cual ellos ocurren. Obtenga expresiones para Q_{i+1}^+ para $i = 1, 2, 3, 4$ en términos de los valores de entrada x_i . Empezando desde el estado de restablecimiento ($Q_0 = 0$), determine las condiciones en las entradas que se originan en una salida $Q^+ = 1$. Cree un nombre apropiado para el circuito.
- c. Dibuje un diagrama de temporización que incluya el reloj y las formas de onda para x , S y Q para verificar sus conclusiones anteriores.
- d. Suponga que se va a usar un flip-flop JK para efectuar la misma función. Dibuje el diagrama del circuito.
17. Un registro de corrimiento de 3 bits utilizando flip-flops D tiene una entrada x y estados de izquierda a derecha Q_0 , Q_1 y Q_2 . Se obtiene una salida z de una compuerta XOR cuyas entradas son x y Q_2 . El principio del diagrama de temporización, que muestra una secuencia de reloj y la entrada x , está dado en la figura P17.

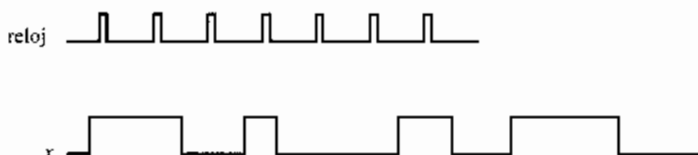


Figura P5

- a. Dibuje el circuito.
- b. Extienda el pulso de reloj en seis periodos más y complete el diagrama de temporización agregando las formas de onda de Q_0 a Q_2 y la salida z .
18. a. Lemon Logic (LL.com) acaba de patentar un nuevo flip-flop Lemon con una entrada, L , y salidas Q y Q' , que tienen las siguientes propiedades: Si $L = 0$, entonces $Q^{n+1} = 0$; si $L = 1$, entonces $Q^{n+1} = (Q^n)'$. Elabore una tabla que muestre los requerimientos de excitación para cada transición de estado. Explique su razonamiento. ¿Hay algo que parezca anómalo?
- b. Otro diseño de Lemon es un cerrojo LM con reloj. Éste se construye conectando la salida de una compuerta XOR a la entrada R de un cerrojo SR con reloj. Una entrada de la XOR se denomina L y también se conecta a la entrada S del cerrojo; la otra entrada se denomina M . Repita la parte a.
19. ¿Por qué los tiempos de establecimiento y de retención de un flip-flop se definen, respectivamente, respecto a el inicio y al final del pulso de reloj?

20. Para el registro de corrimiento que se muestra en la figura 22, deduzca una expresión para el retardo que determinará su máxima frecuencia de reloj. La ecuación debe ser la suma de unos cuantos términos de retardo.
21. Explique por qué el cerrojo maestro-esclavo JK no se considera un flip-flop de activación por flanco. Diseñe un cerrojo maestro-esclavo de tipo D , T o SR que sea activado por flanco.
22.
 - a. Diseñe un registro de carga en paralelo de 4 bits que pueda aceptar sus entradas a partir de dos fuentes diferentes.
 - b. Modifique su diseño de la parte a de manera que sea posible cargar el registro con nuevos valores o mediante la rotación a la derecha.
23. Diseñe un registro de carga en paralelo de 4 bits que pueda cargarse con nuevos valores, rotarse a la izquierda o a la derecha, o que tenga la posibilidad de preservar su valor existente.
24. Diseñe un registro de carga en paralelo de 4 bits que pueda cargarse con nuevos valores o incrementar su valor existente.
25. Diseñe un registro en serie de 4 bits que se pueda cargar o borrar (todo 0) en forma síncrona.
26. Refiérase al diseño del cerrojo SR de la figura 4. Un estudiante curioso quiere conocer lo que resultaría al eliminar los dos inversores en las entradas. El consultor será usted.

Analice el circuito que se produce y construya una tabla de transición para este diseño que es similar a la figura 6. Explique sus resultados.

Capítulo 6

Máquinas secuenciales síncronas

En la introducción del capítulo 5 se presentó un diagrama de bloques de un circuito secuencial. La información se concentró en una parte de este tipo de circuito: los dispositivos de memoria o flip-flops. Aquellos circuitos en los cuales las transiciones de estado se controlan, o sincronizan, mediante un reloj, se denominan circuitos secuenciales *con reloj* o *síncronos*. Existen otros circuitos secuenciales, conocidos como circuitos *asíncronos*, en los cuales las transiciones de estado no las sincroniza un reloj. Éstos son menos comunes, aunque tienen importantes aplicaciones. Pospondremos la exposición de este tipo de circuitos hasta el capítulo 7.¹

Para describir el comportamiento de circuitos lógicos secuenciales, y para poder analizarlos y diseñarlos, se utilizan varias herramientas. En este capítulo introduciremos y formularemos estas herramientas. Se incluyen procedimientos formales para el diseño de máquinas síncronas. Por último, nos concentraremos en una clase de estos circuitos y en su diseño: los llamados *contadores*.

1 CONCEPTOS BÁSICOS

La descripción genérica de un problema que requiere el diseño de un circuito lógico secuencial síncrono puede indicarse de la manera siguiente.

Diseñe un circuito digital cuyas salidas deban tomar valores específicos después de que haya ocurrido una secuencia específica de entradas.

Este enunciado de problema resulta muy amplio, pero es claro que:

- Habrá ciertas secuencias de entradas para el circuito.
- Diferentes secuencias de entradas pasadas constituirán estados o condiciones distintas del circuito.
- Las salidas específicas se producirán únicamente después de una secuencia de entrada especificada.

¹ Con o sin adjetivos que lo califiquen, el término *máquina* se usa a menudo para designar un circuito secuencial, como en el título de este capítulo. Debido a que estos circuitos sólo pueden tener un número finito de estados, reciben también el nombre de *máquinas de estados finitos*. Ya que en el mundo físico sólo pueden existir máquinas de estados finitos, este adjetivo se omite a menudo y los circuitos en cuestión se llaman simplemente *máquinas de estado*. "Máquina" por lo general tiene la connotación de algo físico. Sin embargo, en el uso presente, el término se refiere a una entidad abstracta descrita mediante medios matemáticos, gráficos o tabulares, como se describirá en este capítulo.

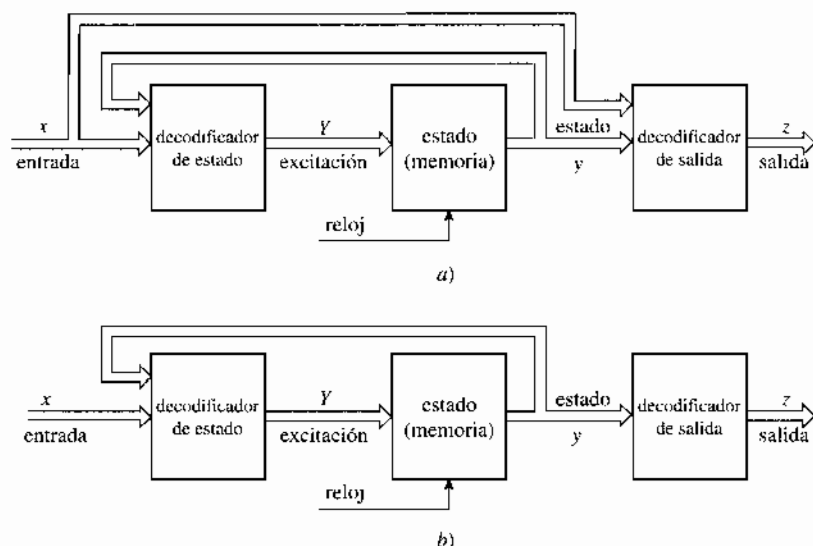


Figura 1. Modelos de máquinas de Mealy y de Moore.

La descripción general no especifica si la salida dependerá exclusivamente de las entradas pasadas o también de la última entrada.

Estas dos posibilidades conducen a diferentes estructuras, las cuales fueron exploradas por Mealy y Moore durante la década de los años 50. Como se mencionó en el capítulo 5, en las *máquinas de Mealy* las salidas dependen tanto del estado presente (resultantes de las entradas pasadas) como de la entrada actual. Las salidas en las *máquinas de Moore* dependen sólo del estado presente (producto de las entradas pasadas). Debe prestarse atención a ambos tipos.

Los diagramas de bloque de los modelos de Mealy y de Moore de máquinas secuenciales se vieron en la figura 1 del capítulo 5. En la figura 1 de este capítulo se presentan versiones más detalladas. Las flechas abiertas implican variables múltiples. Por ejemplo, la entrada x representa el conjunto de variables $\{x_1, x_2, \dots, x_n\}$. La parte combinatoria del circuito se descompone en dos partes independientes: el *decodificador de estado* y el *decodificador de salida*. Los decodificadores de estado en ambos modelos aceptan como entradas tanto entradas primarias (externas) como el estado presente. Sin embargo, en la máquina de Moore, el decodificador de salida sólo acepta el estado presente para producir salidas. En el caso más simple, no hay decodificador de salida en lo absoluto; los propios estados son las salidas. En una máquina determinada, quizás haya algunas salidas que sean del tipo de Moore y otras que no lo sean. Una máquina de estas características debe clasificarse como una de *Mealy*, puesto que al menos *algunas* de sus salidas dependen no sólo del estado, sino también de las entradas. Así, la máquina de Mealy es el tipo más general (y el más común).

El comportamiento de circuitos lógicos secuenciales síncronos puede describirse de diversas maneras. A cualquier pulso de reloj determinado, el estado del circuito es el *estado presente*. Las señales presentes en las terminales de entrada en ese tiempo son las *entradas*. Esta combinación de estado presente y entrada origina dos cosas: una transición al *estado siguiente* y una *salida*. El proceso se repite en el siguiente pulso de reloj, sin embargo, el estado presente es ahora lo que fue el estado siguiente en el pulso de reloj anterior. Es posible concebir este proceso como si nunca terminara; esto es, el "estado siguiente" nunca es uno que se haya encontrado con anterioridad. En este caso, la máquina sería infinita y realmente peculiar. Con excepción de este acontecimiento improbable, en algún lado a lo largo del trayecto, el siguiente estado será uno encontrado previamente. Luego de esto, la máquina repetirá sus pasos una y otra vez, sin que se presente ningún estado nuevo.

Se dispone de varios medios para ilustrar la secuencia siguiente:

estado presente/entrada \rightarrow pulso de reloj \rightarrow estado siguiente/salida

Uno de estos medios es gráfico/diagramático: otro es tabular. Un tercer método utiliza un diagrama similar al diagrama de flujo que describe a un algoritmo. Todos ellos se tratan en este capítulo.

Diagrama de estados

La herramienta gráfica-diagramática para describir el comportamiento de circuitos secuenciales se conoce como *grafo lineal*.² En cada estado del circuito, existe un nodo correspondiente en la gráfica. (El círculo que representa al nodo se hace lo suficientemente grande para que el símbolo correspondiente al estado se pueda escribir en el interior.) Con la máquina en cualquier estado (nodo en la gráfica), a la ocurrencia de un pulso de reloj, existirá una transición de estado al siguiente estado y habrá una salida, ambos de acuerdo con el enunciado del problema. De una máquina de una sola entrada salen dos líneas desde cada nodo, cada una para una entrada 0 y para una entrada 1. En el caso de dos variables de entrada, derivan cuatro líneas desde cada nodo, una para cada combinación de entrada: 00, 01, etc. (¿Cuántas líneas provienen de cada nodo si el número de variables de entrada es n ?) A lo largo de cada línea escribimos el valor de entrada y la salida correspondiente separadas por una diagonal. El grafo resultante recibe el nombre de *diagrama de estados*.

En algunas máquinas de estados se sabe exactamente cuántos estados distintos tiene a partir del enunciado del problema. Sin embargo, en general, el número de estados posibles se desconoce en un principio. Para establecer el diagrama de estados, elegimos de manera arbitraria un estado inicial y lo denominamos, digamos, A. (Los nombres de estado pueden elegirse según resulte conveniente.) Un estado se identifica especificando con exactitud cómo se alcanza. Decir, por ejemplo, que el "estado S se alcanza cuando la entrada es 1" resulta inadecuado.³ Debido a que el enunciado no lo identifica de modo *inambiguo*: ¿Este 1 sigue a otro 1? ¿Es el primer 1 después de una secuencia de 0s? ¿Este 1 sigue a una secuencia de dos o más 1s anteriores? Una especificación precisa sería: "El estado S se alcanza mediante el segundo de dos 1s de entrada después de uno o más 0s".

Ya que es difícil describir, en abstracto, tanto el diagrama de estados como la herramienta tabular que se va a explicar a continuación, seguiremos la exposición con un ejemplo.

EJEMPLO 1

Se va a diseñar una máquina secuencial síncrona con una línea de entrada única x y una sola línea de salida z . Las especificaciones son: la salida será $z = 1$ si y sólo si la secuencia de entrada específica ...0110 ocurre en pulsos de reloj consecutivos; en otro caso $z = 0$. (A menos que otra cosa se establezca, el bit más reciente de una secuencia es aquél a la derecha en todos los casos.) Suponga, por ejemplo, que una secuencia de entrada es 0110110. Los primeros 4 bits identificados constituyen una secuencia de entrada *aceptable* (esto es, que *produce una salida*). Pero el cuarto bit inicia otra secuencia aceptable que se traslapa con la primera. Por consiguiente, la secuencia de salida será ...0001001. (Confírmelo.) Una máquina de estas características recibe el nombre de *detector de secuencia*.

² Un grafo lineal es un conjunto de *nodos*, o *vértices* (dibujados como círculos), interconectados por un conjunto de líneas (o arcos) dirigidas, esto es, líneas que tienen una orientación indicada por una punta de flecha.

³ Con excepción del caso trivial que se describirá más adelante.

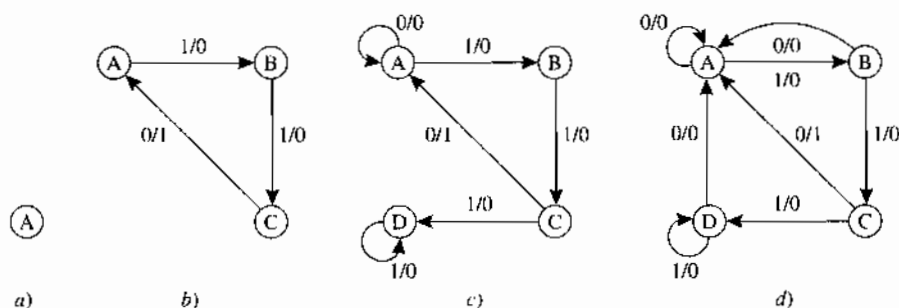


Figura 2. Diagrama de estados del detector de secuencia.

Vamos a identificar el estado inicial A como el que se alcanza por medio de una entrada $x = 0$ independientemente de la secuencia de entradas anterior. No importa si el bit de entrada precedente es 1 o 0 para que un bit de entrada 0 inicie una secuencia aceptable. Empezamos el diagrama de estados dibujando un nodo marcado A. Luego de esto, hay dos estrategias posibles:

- Podemos explorar las consecuencias (estado siguiente/salida) resultado de cada entrada posible que empieza en este estado, y continuar de este modo con todos los estados siguientes que se encuentren a lo largo del camino.
- Podemos suponer una secuencia aceptable y seguir las consecuencias (secuencia de estados siguientes y salida), agregando estados según sea necesario. Después regresamos a cada estado encontrado a lo largo del camino y cubrimos las consecuencias de entradas que no son parte de una secuencia aceptable.

En la figura 2 mostramos el segundo método. Empezando en el estado A (alcanzado por una entrada 0), suponemos una secuencia de entrada 110 para completar una secuencia aceptable. El resultado se muestra en la figura 2b. (Confirme que los dos estados adicionales que se indican deben introducirse durante el proceso. Compruebe también cada uno de los pasos que se siguen.)

Iniciando en cada etapa de la figura 2b, únicamente una de las dos posibles entradas se ha utilizado hasta ahora. A continuación cubrimos las otras posibilidades. A partir del estado A, una entrada de 0 conduce de regreso al estado A/salida 0. Desde el estado B, una entrada de 0 conduce de regreso también a A/salida 0. ¿Pero cuál es el estado siguiente si hay una entrada de 1 mientras la máquina se encuentra en el estado C (el cual se alcanzó mediante una secuencia de entrada 011)? Éste no puede ser alguno de los tres estados alcanzados hasta ahora (confírmelo), por lo que debe corresponder a un estado nuevo D/salida 0. El diagrama de estados hasta ahora se muestra en la figura 2c. Por último, a partir de este nuevo estado, una entrada de 0 lleva de regreso a A/salida 0 y una entrada de 1 conduce de regreso a D/salida 0. El diagrama final se ilustra en la figura 2d.

Estudie el último diagrama. A es el estado que se alcanza mediante el primer bit en una secuencia aceptable. La secuencia se aborta si la siguiente entrada es también 0. Entonces es este último 0 el que empieza una secuencia aceptable. Cualquier número de entradas adicionales de 0 conduce al mismo resultado: el último 0 se convierte en el primer bit de una secuencia aceptable. El último estado D, representa un estado de desperdicio de secuencia aceptable; se llega a él mediante una entrada de 1 siguiendo una secuencia ...011. ■

Advierta en la figura 2 que todos los arcos en el diagrama con excepción de 1 se marcan con una salida de 0. Podría evitarse mucho desorden si adoptamos la convención de que sólo se mostrarán de manera explícita las salidas con valor 1. Después de la salida asociada con un arco particular tome el valor 0 (o 00, 000, etc. para más variables de salida), ya no se indicará explícitamente en el diagrama de estados.

PS	NS, z	
	$x = 0$	$x = 1$
A	A,0	B,0
B	A,0	C,0
C	A,1	D,0
D	A,0	D,0

Figura 3. Tabla de estados del detector de secuencia del ejemplo 1 obtenido a partir de su diagrama de estados.

Al construir un diagrama de estados, existen por lo general dos puntos de decisión principales:

1. Elegir el estado inicial.
2. En un estado particular, decidir si la transición resultante de una entrada particular es para un estado existente o para un estado nuevo aún no identificado.

En algunas máquinas secuenciales (no en todas) existe un estado de *restablecimiento* específico; la máquina debe estar en este estado en el tiempo de inicio. En tales casos, el estado inicial se encuentra predeterminado. Cuando no hay estado de restablecimiento, el estado inicial se elige de manera arbitraria, como en el ejemplo anterior. Si bien el enunciado del problema puede guiar la elección del estado inicial, diferentes diseñadores podrían optar por estados iniciales distintos.⁴ No hay problema. Suponiendo que no hay errores, dos diagramas de estados construidos con estados iniciales diferentes serán *isomórficos*; esto es, se volverán idénticos mediante un intercambio apropiado de nombres de estado.⁵

En cuanto al segundo punto de decisión, una transición a un nuevo estado y no a un estado existente originará más estados en el diagrama de estados. A la larga, un circuito debe implementarse.

Hablando en general, más estados equivalen a mayor número de flip-flops, aunque no proporcionalmente. Un circuito con n flip-flops, por ejemplo, tendrá 2^n estados. Inversamente, entonces, 8 (2^3) estados requerirán tres flip-flops. Sin embargo, incluso un número pequeño, como cinco estados continuarán requiriendo tres flip-flops. Así, si un diagrama de estados ya tiene cinco estados, incrementar el número de éstos hasta ocho no aumentará el número de flip-flops que se necesitan. Entonces, la introducción de más estados en un diagrama de estados quizá simplemente equivalga a introducir redundancias, las cuales podrían eliminarse después.

A pesar de que en el ejemplo anterior construimos un diagrama de estados que describe todas las transiciones de estado de la máquina deseada, todavía no completamos un diseño. Antes de embarcarnos en esta tarea, volveremos a la herramienta tabular para describir el comportamiento de una máquina secuencial.⁶

Tabla de estados

Una tabla de estados se describe mediante sus encabezados (o nombres) de renglón y sus encabezados de columna. Sus entradas ocurren en las intersecciones de los renglones y columnas. Al describir la operación de una máquina síncrona, es usual elegir los encabezados de renglón como los estados presentes y los encabezados de columna como las entradas.

⁴ También podrían darse nombres diferentes a los estados, la ubicación de los nodos podría no ser igual y las curvaturas de las líneas que unen los nodos podrían diferir. Todos éstos son aspectos triviales.

⁵ Esto supone que no se introducirán estados adicionales, como se describirá más adelante.

⁶ Suponga que reconsideramos el enunciado "El estado S se alcanza cuando la entrada es 1". El diagrama de estados consiste entonces en sólo dos estados: S es el estado que se alcanza mediante una entrada 1 y, digamos, T se alcanza por medio de una entrada 0. Cualesquiera otras entradas 0 mientras esté en el estado T regresarán la máquina al estado T. Una entrada 1 enviará a la máquina de regreso al estado S. Cualesquiera otras entradas 1 mantendrán la máquina en el estado S. Construya usted mismo un diagrama de estados. Ésta es una "máquina" trivial.

Puesto que se originan dos resultados (estado siguiente, salida) de una entrada al circuito cuando éste se encuentra en un estado particular, es concebible construir dos tablas independientes. Las entradas en una de ellas serían las salidas del circuito —por lo que ésta recibe el nombre de *tabla de salidas*. Las entradas en la otra tabla corresponderían a los estados siguientes. Puesto que el estado tiene por objetivo mostrar las transiciones de un estado presente al que sigue, podría resultar tentador nombrarla tabla de transición. Pero en el capítulo anterior, se asignó el nombre de tabla de transición de estados a la que especifica el estado siguiente producto de las entradas a un flip-flop para cada estado presente de este mismo dispositivo. Por ello recurrimos a un nombre diferente, que será *tabla de estados*. En el tratamiento presente, el circuito no está limitado a un solo flip-flop sino que abarca una máquina completa.

Recuerde de la figura 1b que la salida de una máquina de Moore sólo depende del estado presente. En una máquina de este tipo, únicamente habrá una combinación de salidas para cada combinación de entrada; en consecuencia, tiene más sentido una tabla de salidas independiente. En una máquina de Mealy, por otro lado, combinaremos las tablas de estados y salidas en una sola en la que las entradas son tanto los estados siguientes como las salidas que se producen, separadas por medio de una coma.⁷ Esto se ilustrará en el ejemplo 1, cuyo diagrama de estados se obtuvo en la figura 2. (Desde luego, una vez que se cuente con esta tabla, será posible separar la parte del siguiente estado y la de la salida en dos tablas independientes si hay alguna razón para hacerlo.)

Construcción de una tabla de estados a partir de un diagrama de estados

Una vez que se dispone de un diagrama de estados, la tabla estado/salida correspondiente se construye con facilidad. El diagrama de estados de la figura 2 tiene cuatro estados. Así que la tabla estado/salida correspondiente contará con cuatro renglones. Empezando en cualquier estado, la salida y el estado siguiente pueden obtenerse del diagrama y anotarse en la tabla. El resultado se indica en la figura 3.

Ejercicio 1. Advierta en la figura 2 que el estado A se alcanza a partir de cada estado (incluso A) mediante una entrada 0 pero con salidas diferentes: 0 a partir de los estados A, B, D y 1 a partir del estado C. En lugar de eso, vamos a suponer que el estado A en el diagrama de estados de la figura 2 se identifica como el alcanzado por una entrada 0 que se origina de una salida 1. Empezamos ahora a partir de la figura 2a y suponga que mientras se está en el estado A o en el B una entrada 0 conduce a un nuevo estado E en vez de regresar a A. Complete el diagrama de estados que resulta. (No necesita seguir el siguiente consejo; no se enredará al cruzar líneas si pone A, B y C en un renglón, con D bajo C y E bajo B.) Después de esto, a partir del diagrama, construya una tabla de estados. ♦

Respuesta⁸

Examine los estados A y E en su tabla del ejercicio 1. Ambos se alcanzan mediante un bit de entrada 0. Por consiguiente, partiendo de cualquier estado, los estados siguientes y las salidas deben ser las mismas para cada bit de entrada. En este caso, estos dos estados no pueden distinguirse uno del otro. Lo anterior constituye la base para una definición:

⁷ En ocasiones también se denomina *tabla de flujo*.

⁸ En este caso, sólo un arco de la gráfica con entrada 1 ingresa a A (de C/salida 1). Cuatro arcos entran a E (de A, B, D y E, todas con 0 de entrada, 0 de salida). D es un estado de desperdicio de secuencia, al que se entra por medio de una secuencia de tres 1s empezando en A; cualesquiera 1s adicionales mientras se está en el estado D mantendrán el estado en D. Una entrada 1 mientras se está en el estado E representa el primer 1 después de uno o más 0s; a ese respecto, una entrada 1 mientras se está en el estado E debe conducir al mismo estado que en el caso de un 1 mientras se está en el estado A, a saber, estado B.

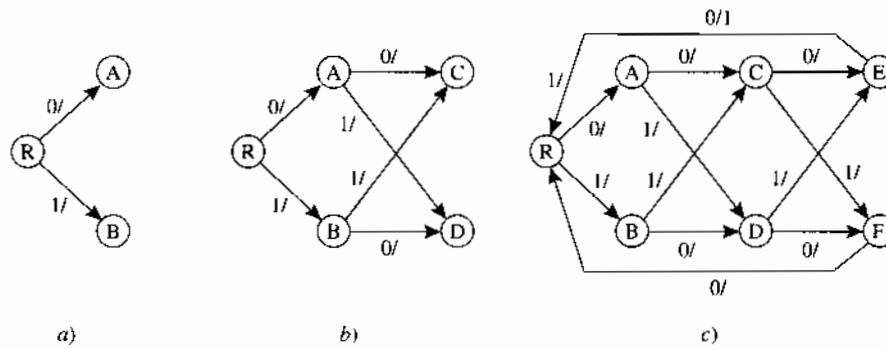


Figura 4. Diagrama de estados del generador de bit de paridad.

Se dice que dos estados serán indistinguibles si, para cada combinación de entrada, las salidas resultantes y los estados siguientes son los mismos.

En realidad, los estados siguientes no necesitan ser los mismos, sólo indistinguibles, como acaba de definirse.

Con esta base, los estados A y E son indistinguibles. Si todos los estados siguientes E en la tabla se sustituyen por A, y se elimina el renglón E, la tabla que se obtuvo en el ejercicio 1 se volverá igual a la que se encontró en la figura 3.

Existen procedimientos formales para extender el concepto recién definido. Seguiremos esta generalización en la sección siguiente y explicaremos formas de reducir el número de renglones de una tabla de estados. En consecuencia, al construir un diagrama de estados en el diseño de un circuito secuencial no hay por qué preocuparse de introducir estados redundantes. Estos estados pueden eliminarse de manera subsecuente. Por otro lado, no hay motivo para extender innecesariamente una tabla de estados, puesto que después se requerirá esfuerzo para reducirla. Cuando haya duda mientras se construya una tabla de estados, introduzca por todos los medios un nuevo estado. Sin embargo, restrínjase si tiene la certeza de que ya se han identificado condiciones relevantes mediante un estado existente.

EJEMPLO 2

(Nota: Uno de los temas tratados en la sección 4 del capítulo 1 es el código de Hamming. Repáselo si necesita recordarlo.) Para un mensaje de n bits, se agregan k bits adicionales, haciendo que la paridad de la secuencia de $(n + k)$ bit resultante sea par o impar (según sea nuestra elección). En este ejemplo, sea $n = 3$ y $k = 1$, elegiremos paridad impar. Suponga que se va a recibir una secuencia de 4 bits. Los primeros 3 bits constituyen el mensaje y el cuarto bit siempre es 0. Si el número de 1s en el mensaje de 3 bits es impar, el bit de paridad continuará siendo 0. Si es par el número de 1s, se generará un bit e insertar en la cuarta posición para volver impar la paridad de la secuencia de 4 bits. En cualquier caso, después del cuarto bit, la transición corresponderá a un estado de restablecimiento, en el cual la máquina está lista para recibir la siguiente secuencia del mensaje. Una máquina de este tipo es un *generador del bit de paridad*. Nuestro objetivo es crear la tabla de estados correspondiente.

El primer paso es crear un diagrama de estados. La máquina estará en el estado de restablecimiento (digamos R) cuando llegue el primer bit. El diagrama de estados parcial después del primer bit se ilustra en la figura 4a. Ahí no puede haber una salida 1 hasta que el bit de paridad llegue y complete una secuencia de 4 bits cuya paridad es par. En cada entrada, la paridad de los

PS	NS,z	
	x = 0	x = 1
R	A,0	B,0
A	C,0	D,0
B	D,0	C,0
C	E,0	F,0
D	F,0	E,0
E	R,1	R,×
F	R,0	R,×

Figura 5. Tabla de estados para el generador de bit de paridad.

bits de entrada hasta ese punto es par o impar, por lo que la transición es uno de los dos siguientes estados posibles: un estado de paridad par o un estado de paridad impar.

El diagrama parcial después del segundo bit de mensaje se muestra en la figura 4b. ¿Por qué es necesario que el siguiente estado correspondiente al segundo bit de entrada sea un nuevo estado en vez de uno de los estados existentes de paridad impar o paridad par, A o B? (Pien-se antes de mirar el pie de página.⁹) Confirme los detalles del diagrama completo que se presenta en la figura 4c. Indique, por ejemplo, todas las secuencias de entradas posibles mediante las cuales se llega a los estados E y F. Una salida de 1 ocurre sólo después de regresar al estado de restablecimiento a partir del estado E que sigue a una de las secuencias 0000, 0110 y 1100.

El siguiente paso consiste en construir la tabla de flujo a partir del diagrama de estados. Éste se muestra en la figura 5. El cuarto bit de entrada nunca es 1; ¿cuál será entonces la salida para $x = 1$ a partir de los estados E y F? Confirme todos los detalles de esta tabla haciendo referencia al diagrama de estados. ■

2 ASIGNACIONES DE ESTADO

En la sección anterior se describió la etapa inicial en el diseño de una máquina secuencial síncrona de modelo Mealy. Según una descripción en palabras de las especificaciones del problema, éste consiste en la construcción de una tabla de estados/salida (posiblemente después de elaborar un diagrama de estados). Durante este proceso, es posible introducir estados redundantes; en consecuencia, la tabla podría contar con más estados que los necesarios para efectuar la tarea que se desea. Los procedimientos para eliminar estados redundantes resultarían de utilidad, pues una se tendría reducida con menos estados que su equivalente, de cierta forma, a la tabla original. Afortunadamente, existen tales procedimientos y los analizaremos después. Por ahora, suponga que se cuenta con una tabla reducida. El estado de una máquina secuencial en un tiempo determinado es la condición en la cual la habían dejado las entradas pasadas. Esta información se almacena en los flip-flops; el estado es, entonces, descrito de manera colectiva por las salidas de los flip-flops. Así, el siguiente paso en el diseño consiste en identificar los estados en la tabla con salidas de flip-flop específicas. Éste es el tema de interés en esta sección. Lo desarrollaremos con referencia a los ejemplos en la sección anterior.

⁹ Si es posible alcanzar A tanto después de un 0 único como de una secuencia 00, por ejemplo, entonces se pierde el conteo de número de bits de entrada. Por consiguiente, no podemos indicar cuando ha llegado el tercer bit para decidir si se genera o no un bit 1 ni podemos decir cuando ha llegado el cuarto bit de manera que regresemos al estado de restablecimiento.

PS	NS,z		$(y_1y_2)^*$			y_1^*			y_2^*		
	$x = 0 \quad x = 1$		y_1y_2	$x = 0 \quad x = 1$		y_1y_2	$x = 0 \quad x = 1$		y_1y_2	$x = 0 \quad x = 1$	
	$x = 0$	$x = 1$									
A	A,0	B,0	A→00	00	01	00	0	0	00	0	1
B	A,0	C,0	B→01	00	10	01	0	1	01	0	0
C	A,1	D,0	D→11	00	11	11	0	1	11	0	1
D	A,0	D,0	C→10	00	11	10	0	1	10	0	1
	a)		b)			c)			d)		

Figura 6. Tablas de estados y transición para el detector de secuencia.

EJEMPLO 3

La tabla de estados que se obtuvo para el detector de secuencia del ejemplo 1 se presentó en la figura 3 y se repite en la figura 6a. El número mínimo de variables de estado necesarias para una implementación de circuito de esta tabla es $\lceil \log_2 4 \rceil = 2$, donde $\lceil k \rceil$ denota el valor *tope* de k , esto es, el entero más pequeño no menor que k . Denominemos las variables de estado como y_1 y y_2 . Hay cuatro combinaciones posibles de valores de estas dos variables. ¿Cómo deben asignarse individualmente estas cuatro combinaciones a cada uno de los cuatro estados? Antes de considerar una respuesta general a esta pregunta, realizaremos la asignación arbitraria que se muestra en la figura 6b. (Temporalmente ignoraremos la salida z y nos concentraremos en los estados.) El resultado es una tabla que, para cada combinación presente de valores de variable de estado y cada valor de entrada, especifica la siguiente combinación de valores de variables de estado. Ésta constituye una *tabla de transición de estados*.

Advierta que hay dos órdenes diferentes: listar los estados en la tabla de estados (alfabético) y las combinaciones de los valores de las variables de estado. Si la asignación se efectúa de manera que se mantengan ambos órdenes, no hay problema. En cualquier otra asignación, es posible sostener uno u otro orden, pero no ambos. Resulta mucho más conveniente mantener el orden de las combinaciones de valores que el orden de los nombres de los estados, ya que el primero se transfiere directamente a un mapa lógico. Esto es lo que se ha hecho en la figura 6b; los estados C y D están fuera del orden alfabético, pero las combinaciones de valores se encuentran en el orden del mapa lógico.

Por simplicidad, la tabla de transición en la figura 6b se divide en dos tablas en las figuras 6c y 6d, una para cada variable de estado. Si fuéramos a implementar el diseño con flip-flops D , cada una de estas tablas representaría el mapa de excitación para uno de los flip-flops; para un flip-flop D , la entrada presente (excitación) es la misma que el estado siguiente. Sólo por razones pedagógicas, lo implementaremos con flip-flops JK .¹⁰

El primer requerimiento corresponde a determinar mapas lógicos para las excitaciones J y K relativas a cada flip-flop. Para esto recurrimos a los requerimientos de excitación para flip-flops JK dados en la figura 17 del capítulo 5 y repetidos aquí en la figura 7a. Para cada flip-flop, esta tabla indica los valores requeridos de J y K correspondientes a cada transición desde un valor de estado presente hasta el siguiente valor de estado. En cada caso, J y K se obtendrán como la salida de un circuito combinatorio cuyas entradas son la *entrada* x del circuito y los estados presentes y_1 y y_2 . Esto implica combinar las transiciones desde el estado presente al siguiente en las figuras 6c y 6d con la tabla de requerimientos de transición de la figura 7a. Así, a partir del

¹⁰ Sólo se necesita un circuito lógico combinatorio para la excitación cuando la implementación se lleva a cabo con flip-flops D . Con flip-flops JK , se necesitan dos circuitos, cada uno para J y K . Resulta posible, desde luego, que el circuito requerido con un flip-flop D sea más complejo; sin embargo, con las implementaciones de la actualidad a partir de PLD suele preferirse utilizar flip-flops D . Además, los flip-flops JK requieren más área de CI en la tecnología ASIC.

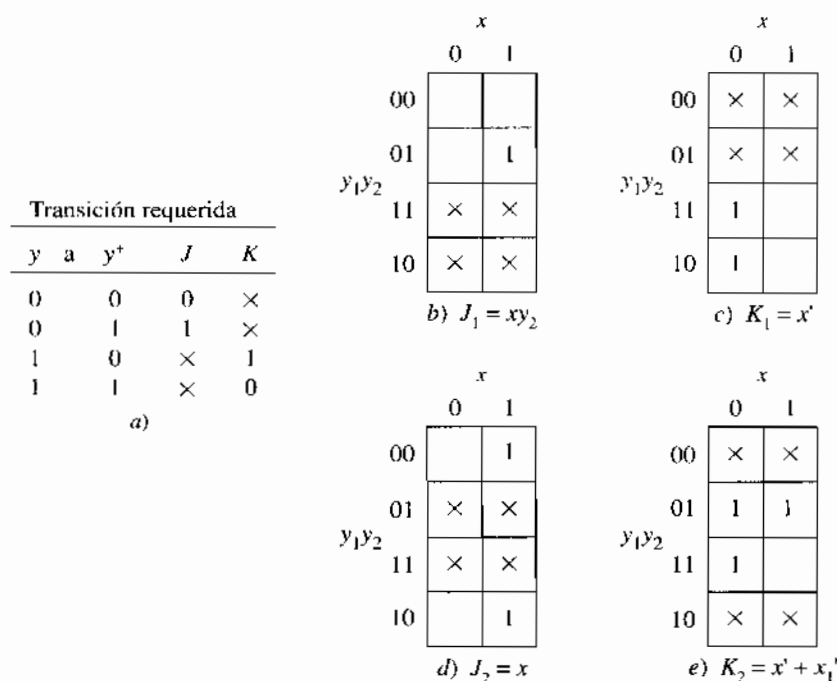


Figura 7. Mapa de excitación para el detector de secuencia.

estado $y_1y_2 = 11$ y $x = 1$, la transición es a $y_1^+y_2^+ = 11$ de acuerdo con las figuras 6c y 6d. Esto es, tanto para y_1 como para y_2 , la transición es de 1 a 1 para $x = 1$. Pero según la figura 7a, los requerimientos para una transición de 1 a 1 son $J = x$ y $K = 0$. Éstos constituyen el contenido de una celda o cuadrado en los mapas lógicos para cada J y K . Los mapas lógicos complementados se muestran en las figuras de la 7b a la 7e. (Confirme cada uno de ellos, así como las expresiones J y K dadas bajo los mapas.)

Para completar la implementación, debe obtenerse una expresión para la función de salida. De acuerdo con la tabla de flujo en la figura 6a, la salida es 1 para exactamente un estado (estado C) y una entrada ($x = 0$). Puesto que C tiene la asignación $y_1y_2 = 10$, la expresión para la función de salida es

$$z = x' y_1 y_2'$$

La implementación completa se muestra en la figura 8. (Confírmela totalmente.) ■

Análisis

Una vez que se diseña un circuito secuencial y se construye un diagrama lógico, ¿cómo podemos indicar que no se han cometido errores y que la salida del circuito satisface realmente las especificaciones originales? Según la explicación de los circuitos combinatorios en el capítulo 3, se lleva a cabo un proceso denominado *verificación*. Éste implica efectuar mediciones (de voltaje, digamos) en puntos apropiados en cualquier circuito (no sólo circuitos lógicos) para verificar que los valores reales son los que se suponen teóricamente.

También en este caso, como se explicó en el capítulo 3, no hay razón para implementar físicamente el circuito antes de la verificación. Una vez que se ha obtenido un circuito secuencial en papel (o generado por software), es posible *analizar* el circuito en diversos puntos para verificar que los valores lógicos en ellos sean en realidad los requeridos por las especificaciones de

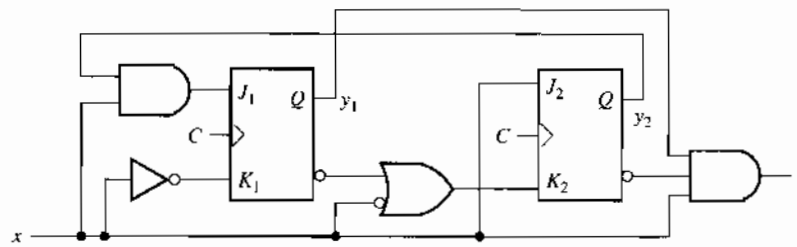


Figura 8. Implementación del detector de secuencia.

		Tiempo										
x		0	1	1	0	1	1	0	0	1	1	1
J_1	xy_2	0	0	1	0	0	1	0	0			
K_1	x'	1	0	0	1	0	0	1	1			
J_2	x	0	1	1	0	1	1	0	0			
K_2	$(xy_2)'$	1	1	1	1	1	1	1	1			
y_1^+		0	0	1	0	0	1	0	0			
y_2^+		0	1	0	0	1	0	0	0			
z	$x'y_1y_2'$	0	0	0	1	0	0	1	0			

Figura 9. Tabla de temporización para la implementación de la figura 8.

diseño. Comparado con el proceso de diseño, el análisis de circuitos lógicos resulta bastante trivial. Se empieza en cualquier punto en el circuito (compuerta o salidas de circuito, o MUX, flip-flop o entradas de registro) y se determinan las expresiones lógicas para estas variables. Lo anterior se repite hasta que se obtienen expresiones para todas las salidas. Después se insertan todos los valores de entrada posibles en estas expresiones. Los valores que resultan se comparan con los supuestos.

Como ejemplo, volvamos al diseño del detector de secuencias de la figura 8. Efectúe un análisis del circuito y obtenga expresiones para las entradas J y K y la salida z . (No vea la figura 9 hasta que haya terminado el análisis.) Las líneas sobre el eje de tiempo en la figura 9 representan el flanco ascendente de la señal de reloj. Utilizando estas expresiones y la tabla de transición de flip-flops JK , y suponiendo la secuencia de entrada que se indica en la primera línea, los valores de las otras variables se determinan columna por columna. Advierta que, cuando $x = 0$, los valores de las entradas J y K no dependen de los estados de los flip-flops; por consiguiente, los siguientes estados en la primera columna se basan sólo en $x = 0$. Cuando $x = 1$, los valores de J_1 y K_2 dependen de los estados (no de los siguientes estados en la misma columna, sino de los estados en la previa). Verifique el resto de las columnas y complete las últimas tres en la figura 9. Compruebe que las salidas cumplen las especificaciones.

Ejercicio 2. Utilizando la información en la figura 9, elija una escala apropiada y dibuje un diagrama de temporización que incluya la entrada del circuito, las entradas de flip-flop, los estados siguientes y la salida del circuito. ♦

Reglas prácticas para asignar estados

Existen varios cabos sueltos en la formulación anterior. El primero es la simple observación de que, cuando se obtiene una tabla de transición después de que se realiza una asignación de estados, como en la figura 6b, no es esencial reescribirla en la forma de las tablas de transición de variables de estado individuales, como se hizo en la figura 6c y 6d, antes de construir los mapas de excitación. En lugar de eso, en cada valor de entrada, concéntrese en la columna correspondiente a una de las variables de estado, digamos y_1 , bloqueando mentalmente las otras de su percepción, y construya los mapas de manera directa a partir de la tabla de transición general. Practique con la figura 6b.

Una consideración más importante es la siguiente. Dada una tabla con k estados, el número de variables de estado necesario para implementarla es $n = \lceil \log_2 k \rceil$. Se necesita una decisión inmediata en cuanto a cuál de las 2^n combinaciones de valores de variables de estado deben asignarse a cada uno de los k estados. Para un número de estados no trivial, existen muchas posibilidades diferentes para esta asignación.

Ejercicio 3. La tabla de estados para el detector de secuencias en el ejemplo 1 se proporcionó en la figura 6a. La implementación del circuito utilizando la asignación de la figura 6b se señaló en la figura 8. En lugar de eso, utilice la siguiente asignación y encuentre una implementación para el circuito: A: 00, B: 01, C: 11, D: 10. Compare el número de compuertas con el número en la figura 8. ♦

Respuesta¹¹

En general, la elección de la asignación influirá en la implementación. Asignaciones diferentes conducen a mapas diferentes de excitación y salida de flip-flop y, por consiguiente, a expresiones distintas para las funciones de excitación y de salida. Desafortunadamente, no hay teoría general acerca de las asignaciones —y por ello ningún algoritmo— que permitan determinar la implementación más simple. La experiencia es la única guía para efectuar una asignación de estados. Los modelos generales de circuitos secuenciales se presentaron en la figura 1. Aunque el circuito real de la figura 8 es bastante simple, ilustra bien el modelo Mealy. La parte no secuencial del circuito consiste en dos clases. El circuito combinatorio que implementa las excitaciones y el que implementa la salida, como se esperaba. Luego de efectuar una reducción de estados (hasta que usted aprenda cómo, tendrá que subcontratar esta tarea), se fija la extensión de la memoria (el número de flip-flop). La economía de implementación, entonces, tiene que ver con la reducción del número de CI (y compuertas) para el decodificador de estado, el decodificador de salida o ambos.

Recuerde que el número de implicantes primos y el número de literales en un implicante primo pueden reducirse cuando hay muchos minitérminos adyacentes. Por tanto, esto puede reducirse a lo siguiente: ¿cómo elegimos asignaciones de estado de modo que se alcance un gran número de adyacencias? No es posible deducir mucho en aras de las generalidades a partir del análisis del ejemplo 3. Sin embargo, con base en una gran experiencia, se han formulado algunas “reglas” heurísticas como guías al realizar una asignación de estado para el caso de una sola entrada. La figura 10 lista varias de estas reglas, en orden de prioridad.

Para una tabla de estado determinada, es improbable que puedan obtenerse todas las adyacencias especificadas por estas reglas. Cuando se presenta un conflicto, se aplica primero la regla de mayor prioridad. Incluso si las reglas se implementan por completo, eso no garantiza una asignación óptima. Esto es, las reglas no conforman un algoritmo óptimo.

¹¹ Una compuerta más que el número en la figura 8.

1. A dos estados presentes deben asignarse códigos adyacentes si éstos tienen el mismo estado siguiente para:
 - a. Cada combinación de entradas.
 - b. Diferentes combinaciones de entradas, si el estado siguiente también puede proporcionar asignaciones adyacentes.
 - c. Algunas combinaciones de entradas, aunque no necesariamente todas.
2. Para todas las entradas, los códigos asignados a los estados siguientes para cada estado presente deben ser adyacentes.
3. Las asignaciones deben simplificar la función de salida.

Figura 10. Reglas de asignación de estados.

Además, no conducen necesariamente a una asignación única; lo cual posibilita que las adyacencias requeridas se consigan por medio de asignaciones diferentes. Incluso así, las reglas reducirán el número de alternativas que deban verificarse. Por último, aun para la misma asignación, el número de compuertas lógicas utilizando flip-flops *JK* podría ser diferente del número que utiliza flip-flops *D* o de otro tipo. A pesar de todo eso, es razonable el uso de estas reglas como guía.

EJEMPLO 4

Una aplicación de las reglas de asignación en la implementación de una máquina secuencial se ilustra en la tabla de estados que se muestra en la figura 11a. Puesto que hay siete estados, el circuito necesitará $\lceil \log_2 7 \rceil = 3$ flip-flops. Las adyacencias de estados demandadas por las reglas de adyacencia se ilustran en la figura 11b. (No deje de confirmarlas.) Después de esto tenemos el problema de determinar una asignación de estado de manera que se consiga el mayor número posible de estas adyacencias. Para ayudar en este proceso, es factible crear un mapa de asignación, como se indica en la figura 11c. Éste es un mapa cuyas coordenadas son las tres variables de estado. Cada celda en el mapa corresponde a una combinación de valores de variables de estado.

La ubicación de los estados en el mapa se inicia decidiendo el estado que tendrá la asignación 000. Si hay un estado de preestablecimiento en el problema, resulta razonable darle esta asignación; si no, la elección es arbitraria. Suponga que la combinación 000 se asigna al estado A; entonces G debe tener una asignación adyacente. Pero cada celda en el mapa tiene otras tres adyacentes a ella; cuál se elige para G depende de qué otras adyacencias se necesiten. En el ejemplo presente, no se requiere que G sea adyacente a cualquier otro estado, por lo que la ubicación resulta muy flexible. Una posibilidad consiste en asignar 001 a G. Las asignaciones restantes se efectúan utilizando el mismo método, como en el mapa de asignación de la figura 11c.

PS	NS,z					
	$x = 0$	$x = 1$				
A	B,1	B,0	Regla	Adyacencias	y ₂ y ₃	y ₁
B	C,0	D,1				
C	E,1	F,0	1a	AG	00	A E
D	F,0	E,1	1b	CD si EF	01	G F
E	G,0	A,0		EF si AG	11	
F	A,0	G,0	2	CD, EF, AG	10	C D
G	B,0	B,0	3	AC, BD		

a)

b)

c)

Figura 11. Tabla de estados de ejemplo, adyacencias y mapa de asignación.

		$y_1'y_2'y_3'$		z		xy_1				xy_1			
		$x=0$	$x=1$	$x=0$	$x=1$								
$y_1y_2y_3$						00	01	11	10	00	01	11	10
A	000	111	111	1	0	1	×	×	1	×	1	1	×
G	001	111	111	0	0	1	×	×	1	×	1	1	×
	011	—	—	—	—	×	×	×	×	×	1		×
C	010	100	101	1	0	×	×	×	×	×			×
E	100	001	000	0	0	1	×	×	1	×			×
F	101	000	001	0	0								
B	111	010	110	0	1								
D	110	10	100	0	1								

a)

$J_1 = 1$

b)

$K_1 = y_2' + x'y_3$

c)

$$J_2 = y_1'$$

$$K_2 = y_3'$$

$$J_3' = (x' + y_1')(x + y_1 + y_2')$$

$$K_3 = (x + y_1)(x' + y_3')$$

$$z = x'y_1'y_3' + xy_1y_2$$

Figura 12. Tabla de transición y mapas de excitación.

El siguiente paso en el proceso de diseño es construir tablas de transición y salida. Esto se efectúa utilizando el mapa de asignación y la tabla de estados dadas. El resultado se ilustra en la figura 12a. Supongamos de nuevo que los flip-flops *JK* se usarán en la implementación; refiérase a la figura 7a para los requerimientos de transición correspondientes a este flip-flop. De acuerdo con las figuras 7a y 12a, construimos los mapas de excitación *J* y *K* para cada flip-flop. El resultado para el primer flip-flop se muestra en la figura 12b. Los otros se obtienen de modo similar. En la figura 12c se proporcionan expresiones para las funciones de excitación y salida. ■

Ejercicio 4. Construya el mapa de salida y los mapas de excitación para los otros dos flip-flops en el ejemplo 4 y confirme las expresiones indicadas en la figura 12c. ♦

Ejercicio 5. En el ejemplo 4, suponga que la implementación recurrirá a flip-flops *D*. Determine los mapas para las excitaciones *D* y, a partir de ellos, el decodificador de estados. Compare el hardware obtenido con el correspondiente al caso que utiliza flip-flops *JK*. ♦

EJEMPLO 5

El objetivo de este ejemplo es determinar un buen esquema de asignación para la tabla de estados de la figura 13a.

El primer paso es determinar las adyacencias utilizando las reglas de adyacencia; éstas se listan en la figura 13b (verifíquelas). Con siete estados, el número de variables de estado necesarias corresponde a tres. Cada celda en un mapa de tres variables es adyacente a otras tres celdas, por lo que cada estado puede ser adyacente, a lo más, a otros tres estados. En un mapa completo de tres variables, se dispone consecuentemente de un total de 12 adyacencias. (Confírmelo.) Únicamente hay siete estados en este ejemplo; ello implica que el número máximo de ad-

PS	NS, z		Regla	Adyacencias	y_1	y_2y_3	y_1	y_2y_3
	x = 0	x = 1						
A	B,0	B,1	1c	AG, BF, CE	00	01	00	01
B	F,0	D,1						
C	E,1	G,1	2	AC, AF, BC	11	10	11	10
D	A,0	C,0						
E	D,1	G,0	3	DF, DG, EG	10	00	01	00
F	F,0	A,0						
G	C,1	B,0						

Figura 13. Tabla de estados, adyacencias requeridas y mapas de asignación para el ejemplo 5.

yacencias presentes es igual a nueve. A partir de la lista de adyacencias requeridas, es claro que se necesita que A sea adyacente a los otros cuatro estados: G, C, F y B. Puesto que la adyacencia de más baja prioridad es AB, ésta debe ser la primera en abandonarse. De manera similar, debe dejarse la adyacencia CG pues C (y G) no puede ser adyacente a los otros cuatro estados.

Siempre que haya una elección, trate de alcanzar esas adyacencias en un nivel de prioridad que también se requiera en un nivel inferior. Además de AB y CG (las cuales abandonamos), de igual modo se requieren las adyacencias restantes en la regla 3 por medio de reglas de orden superior.

El número de adyacencias que pueden obtenerse en este ejemplo viene a ser igual a nueve, igual al máximo posible. La figura 13c muestra dos mapas de asignación; cada uno alcanza las nueve adyacencias requeridas. Utilizando flip-flops JK, la primera puede implementarse con cinco compuertas AND y cuatro OR. La segunda necesita una compuerta OR más. ■

Ejercicio 6. Efectúe las realizaciones para las dos asignaciones indicadas en el ejemplo 5 y confirme los resultados establecidos. Convierta todas las compuertas a NAND. ♦

3 PROCEDIMIENTO DE DISEÑO GENERAL

Cada uno de los elementos de un procedimiento para el diseño de máquinas secuenciales síncronas se ha explicado en las secciones anteriores de este capítulo. Estamos listos ahora para consolidar estos elementos en un procedimiento de diseño general. Ilustraremos éste aplicándolo a algunos ejemplos específicos.

Máquina de Mealy

Los circuitos de Mealy y de Moore son modelos que se mostraron en la figura 1. Cuando un problema de diseño de circuito secuencial se especifica en términos de las salidas deseadas para secuencias de entrada específicas, ningún modelo se especifica de manera general. Para un requerimiento de diseño dado, resulta concebible efectuar el diseño con base en cualquier modelo. Esto quiere decir que es posible obtener dos diseños diferentes (tablas de estados). Significa también que uno de los diseños puede obtenerse del otro. En este libro trataremos tanto con la máquina de Mealy como con la de Moore. El procedimiento de diseño general para las máquinas de Mealy se proporciona en la figura 14.

1. *Tabla de estados*: Dada la especificación de un problema en lenguaje natural, construya una tabla de estados que satisfaga las especificaciones, construyendo quizá primero un diagrama de estados.
2. *Tabla reducida equivalente*: Use procedimientos apropiados para determinar estados equivalentes y eliminar estados redundantes, generando de esa manera una tabla reducida equivalente. (Los procedimientos se considerarán en la sección siguiente.)
3. *Asignación de estados*: Elija una asignación de estados.
4. *Tablas de transición y de salida*: Use la asignación para construir éstas.
5. *Mapas de excitación*: Elija un tipo de flip-flop; utilizando la tabla de transición y los requerimientos de excitación para el flip-flop elegido, construya mapas de excitación.
6. *Funciones de excitación*: Obtenga expresiones para éstas a partir de los mapas.
7. *Funciones de salida*: Obtenga expresiones para éstas a partir de la tabla de salidas.
8. *Implementación*: Implemente el decodificador de estados a partir de las funciones de excitación y el decodificador de salida a partir de las funciones de salida.

Figura 14. Procedimiento de diseño para máquinas de Mealy.

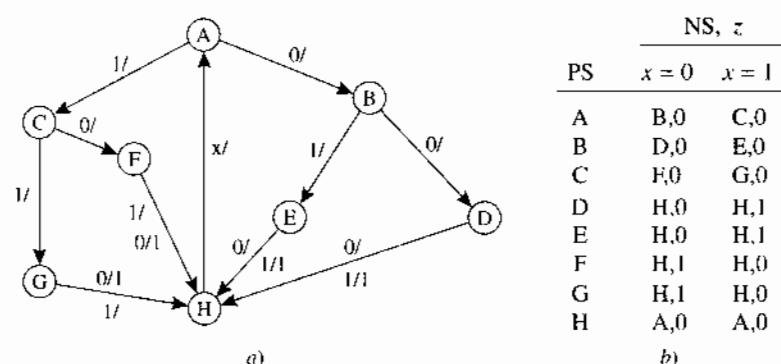


Figura 15. Diagrama de estados y tabla de estados de un detector de cambio de nivel.

EJEMPLO 6

Diseñaremos un circuito secuencial que tiene una sola línea de entrada x y una línea de salida z . Empezando en un estado de restablecimiento, el circuito recibe secuencias de entrada consistentes en palabras binarias de 3 bits. Cada palabra de entrada sigue a la palabra precedente con un retardo de un periodo de reloj. El circuito debe encontrarse en el estado de restablecimiento al principio de cada palabra. La salida será $z = 1$ luego de recibir el tercer bit de una palabra si el número total de cambios de nivel (de 0 a 1 o de 1 a 0) es impar (101, por ejemplo, tiene dos cambios de nivel, en tanto que 001 sólo tiene uno). Un circuito de estas características recibe el nombre de *detector de cambio de nivel*.

Como primer paso, obtendremos el diagrama de estados. Iniciando desde el estado de restablecimiento, sin importar cuál sea el tercer bit, el circuito esperará un periodo de reloj y regresará al estado de restablecimiento al cuarto pulso de reloj. Esto significa que el tercer bit de la palabra de entrada envía el circuito a un estado de *espera*. La figura 15a muestra el diagrama de estados. El estado de restablecimiento es A y el de espera corresponde a H. (Para confirmar este diagrama, cubra el estado de espera y todas las líneas que llegan y salen de él; posteriormente describa cada uno de los estados alcanzados después de 2 bits en términos del número de cambios de nivel. Confirme la salida resultante de cada entrada que envía el circuito al estado de espera.) El diagrama y el enunciado del problema hacen claro que cada estado es único y que no hay estados redundantes; esto es, no existen dos estados que sean equivalentes entre sí.

Reglas	Adyacencias	y_1 0 1	$y_2 y_3$	$y_1' y_2' y_3'$		z			
				$x = 0 \quad x = 1$		$x = 0 \quad x = 1$			
1a	DE, DF, DG	00	A	H	A \rightarrow 000	001	101	0	0
	EF, EG, FG		B	C	B \rightarrow 001	110	111	0	0
2	FG, BC, DE	01	G	E	G \rightarrow 011	100	100	1	0
	DE, FG		F	D	F \rightarrow 010	100	100	1	0
3	DE, FG	11	G	E	H \rightarrow 100	000	000	0	0
			C	D	C \rightarrow 101	010	011	0	0
Adyacencias no alcanzadas: DG, EF.		10	F	D	F \rightarrow 111	100	100	0	1
			D	D	D \rightarrow 110	100	100	0	1

a)

b)

c)

Figura 16. Reglas de adyacencia, mapa de asignación y tabla de transición para el detector de cambio de nivel.

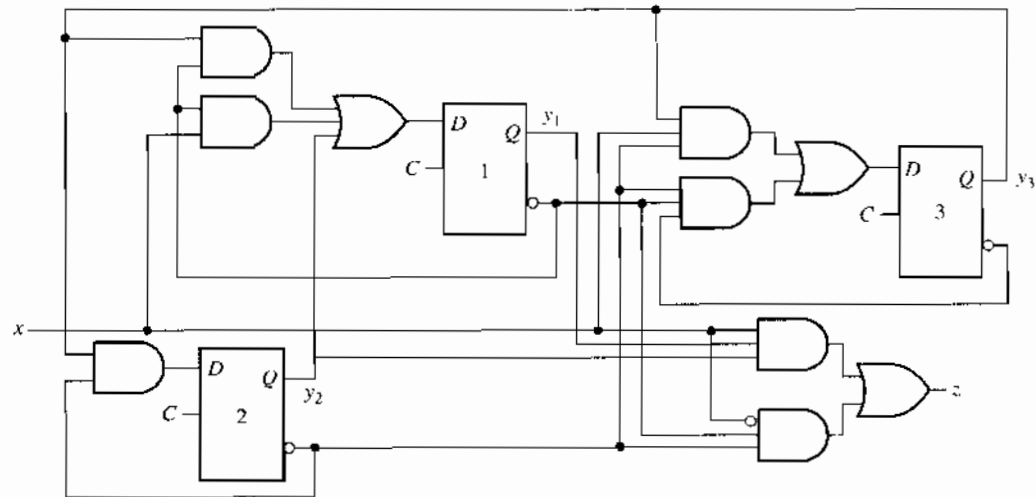


Figura 17. Implementación del detector de cambio de nivel.

El siguiente paso es establecer la tabla; ésta se construye sin dificultades a partir del diagrama de estados y se muestra en la figura 15b. (Confirme esto.) Puesto que cada estado es único, la tabla de estados ya no puede reducirse más. El número de flip-flops que se necesita es $\lceil \log_2 8 \rceil = 3$. Las reglas prácticas para la adyacencia, indicadas en la figura 16b, conducen al mapa de adyacencia de la figura 16b.

Para variar, ahora usaremos flip-flop D , en los cuales la excitación presente es la misma que el estado siguiente. Por lo tanto, las entradas en la tabla de transición especifican también las excitaciones D , de manera que los mapas lógicos para las D pueden construirse en forma directa a partir de las tablas de transición. Lo mismo ocurre para la salida. Daremos los resultados aquí, pero esperamos que usted realice los ejercicios para confirmarlos. Las siguientes expresiones resultan de los mapas.

$$\begin{aligned} D_1 &= y_2 + xy_1' + y_1'y_3 & D_2 &= y_2'y_3 \\ D_3 &= y_1'y_2'y_3' + xy_2'y_3 & z &= x'y_1 + xy_1y_2 \end{aligned}$$

El circuito que se produce se muestra en la figura 17. ■

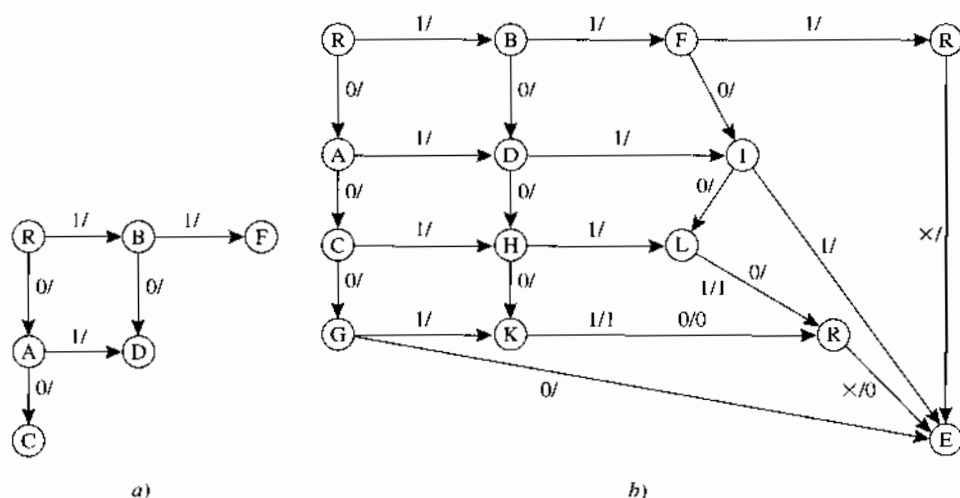


Figura 18. Diagrama de estados del detector de errores en palabras de código 2 de 5.

Ejercicio 7. Utilizando la tabla de transición de la figura 16c, construya los mapas de excitación y salida. A partir de éstos, confirme las expresiones anteriores para las excitaciones D y la salida. Verifique la implementación de la figura 17. ♦

EJEMPLO 7

Una máquina secuencial síncrona tendrá una línea de entrada y una línea de salida. El circuito recibirá mensajes de palabras de 5 bits codificadas en un código 2 de 5 (?). (Véase el capítulo 1 para una descripción de códigos.) El propósito del circuito es detectar un error en cualquiera de las palabras. Así, la salida se convierte en 1 cada vez que una palabra de 5 bits no representa una palabra de código válida. Al final de cada palabra la máquina regresa al estado de restablecimiento.

Diagrama de estado. Considere que el estado de restablecimiento se indica por medio de la letra R. Una posibilidad para la estructura del diagrama de estados es un árbol, que comience en el estado R. En un diagrama de este tipo, cuando la máquina está en cualquier estado, cada uno de los dos valores de entrada posibles conduce a un nuevo estado. Así, el diagrama de estados tendrá $2^5 = 32$ estados. El que no todos éstos sean estados independientes se verá al examinar la información que se necesita luego de recibir el bit k -ésimo:

- ¿Cuántos bits se han recibido hasta ahora?
- ¿Cuántos de éstos bits son 1?

Si se inicia con el estado de restablecimiento, después de la recepción del segundo bit de una palabra, existen tres posibilidades. El número de bits 1 recibidos es 0, 1 o 2. El diagrama de estados parcial se muestra en la figura 18a. Las tres posibilidades identifican exactamente tres estados después de recibir el segundo bit. Una estructura de árbol requeriría cuatro estados en este punto. Luego de recibir el tercer bit de la palabra, hay cuatro posibilidades para el número de bits 1 recibidos hasta ese punto —0, 1, 2 y 3— y consecuentemente cuatro nuevos estados, indicados mediante G, H, I, J en la figura 18b. Advierta que sin importar cuál sea el cuarto bit en el estado presente J, la palabra recibida nunca estará en el código 2 de 5 (?). Por consiguiente, a partir

NS,z			Regla	Adyacencias	$y_1'y_2'y_3'y_4'$		
PS	$x = 0$	$x = 1$			$y_1y_2y_3y_4$	$x = 0$	$x = 1$
R	A,0	B,0	1a	EK, EL, KL			
A	C,0	D,0	1c	GJ, IJ	R → 0000	1100	1110
B	D,0	F,0	2	AB, CD, DF,	E → 0001	0000	0000
C	G,0	H,0		GH, HI, IJ,	K → 0011	0000	0000
D	H,0	I,0		EK, EL, KL	0010	xxxx	xxxx
E	R,1	R,1	3	EK, EL	C → 0100	1011	1010
F	I,0	J,0			L → 0101	0000	0000
G	E,0	K,0			F → 0111	1000	1001
H	K,0	L,0			D → 0110	1010	1000
I	L,0	E,0			I → 1000	0101	0001
J	E,0	E,0			J → 1001	0001	0001
K	R,1	R,0			G → 1011	0001	0011
L	R,0	R,1			H → 1010	0011	0101
					A → 1100	0100	0110
					1101	xxxx	xxxx
					1111	xxxx	xxxx
					B → 1110	0110	0111

Figura 19. Tabla de estados, mapa de asignación y tabla de transición para el ejemplo.

del estado J, el siguiente estado será uno de error (para el cual se reserva la letra E), aunque la salida no se convertirá en 1 hasta el arribo del quinto bit, sea éste un 1 o un 0.

El diagrama de estados terminado se indica en la figura 18b; para evitar el apiñamiento en el diagrama, con líneas que regresen hacia R desde cada uno de los estados alcanzados después de los 4 bits, se proporciona una segunda copia de R cerca de estos últimos estados. Las dos copias de R constituyen el mismo estado.

Tabla de estados. El siguiente paso es construir la tabla de estados a partir del diagrama de estados. Realice esto y verifique la tabla que se da en la figura 19a.

Mapa de asignación. El número de flip-flops que se necesitan es $\lceil \log_2 13 \rceil = 4$. Aplique las reglas prácticas para las adyacencias de estados y confirme la lista que se da en la figura 19. Un mapa de asignación que alcanza al menos una de las adyacencias se proporciona en la figura 19b. Es importante alcanzar el total de las tres adyacencias que requiere la regla 1. Puesto que las reglas de prioridad inferior requieren dos de ellas, éstas dos son las que en realidad se alcanzan. (Confirme todo conforme vaya avanzando.)

Tabla de transición. La tabla de transición que resulta del mapa de adyacencia se muestra en la figura 19c. Puesto que cuatro variables de estado implican 16 estados posibles, y hay en realidad únicamente 13, tres de las combinaciones de variables de estado no corresponden a los estados de la máquina. Por consiguiente, no nos preocupará cuáles serán los siguientes estados resultantes de tales estados presentes no existentes.

Tipo de flip-flop, funciones de excitación y salida. Suponga el empleo de flip-flops D en la implementación. Los mapas de excitación y salida requieren cinco variables. Los mapas de excitación se muestran en la figura 20. Confirme cada uno de estos mapas y las expresiones de excitación y salida resultantes que siguen. (El mapa de salida es tan simple que no se indica.)

$$D_1 = y_1'y_4' + y_1'y_2y_3$$

$$D_2 = y_1y_2 + y_1'y_2'y_4' + x'y_1y_3'y_4' + xy_1y_3y_4'$$

$$D_3 = y_1y_2y_3 + x'y_1y_2'y_4' + xy_1y_2 + xy_1'y_3'y_4' + xy_1y_3y_4$$

		$x = 0$				$x = 1$			
		$y_1 y_2$				$y_1 y_2$			
		00	01	11	10	10	11	01	00
$y_3 y_4$	00	1	1			1	1		
	01			×				×	
	11		1	×			1	×	
	10	×	1			×	1		

D_1

		$x = 0$				$x = 1$			
		$y_1 y_2$				$y_1 y_2$			
		00	01	11	10	10	11	01	00
$y_3 y_4$	00	1		1	1	1		1	
	01			×				×	
	11		1	×				×	
	10	×				×		1	1

D_2

		$x = 0$				$x = 1$			
		$y_1 y_2$				$y_1 y_2$			
		00	01	11	10	10	11	01	00
$y_3 y_4$	00				1	1	1	1	
	01			×				×	
	11			×				×	1
	10	×		1	1	×		1	

D_3

		$x = 0$				$x = 1$			
		$y_1 y_2$				$y_1 y_2$			
		00	01	11	10	10	11	01	00
$y_3 y_4$	00		1		1				1
	01			×	1			×	1
	11			×	1		1	×	1
	10	×			1	×		1	1

D_4

Figura 20. Mapas de excitación para el detector de errores.

$$D_4 = y_1 y_2' + x y_1 y_3 + x y_2 y_3 y_4 + x y_1' y_2 y_3' y_4'$$

$$z = x' y_1' y_2' y_4 + x y_1' y_3' y_4$$

Máquina de Moore

Como se muestra en el modelo de la figura 1b, en una máquina de Moore las salidas no dependen de manera directa de las entradas. En consecuencia, en la tabla de estados de una máquina de Moore hay una sola columna de salidas para cada estado presente, independiente de la entrada. Un ejemplo ilustrará algunas de las características.

EJEMPLO 8

Diseñaremos una máquina de estado síncrona que servirá como un verificador de paridad impar. Las entradas, que serán verificadas respecto a la paridad impar, llegan a una línea de entrada x , aunque la paridad se verifica sólo mientras la señal en otra línea de entrada y (entrada de sincronía) es 1. Una salida z , dependiente sólo del estado, se convierte en 1 cuando la paridad no es impar. Una posible secuencia de entradas y salidas es:

y : 0 0 0 1 1 1 1 1 1 1 1 1 0 0
 x : × × × 1 0 1 0 1 1 1 0 1 0 0 × ×
 z : - - - 0 0 0 0 1 1 1 1 0 - - -

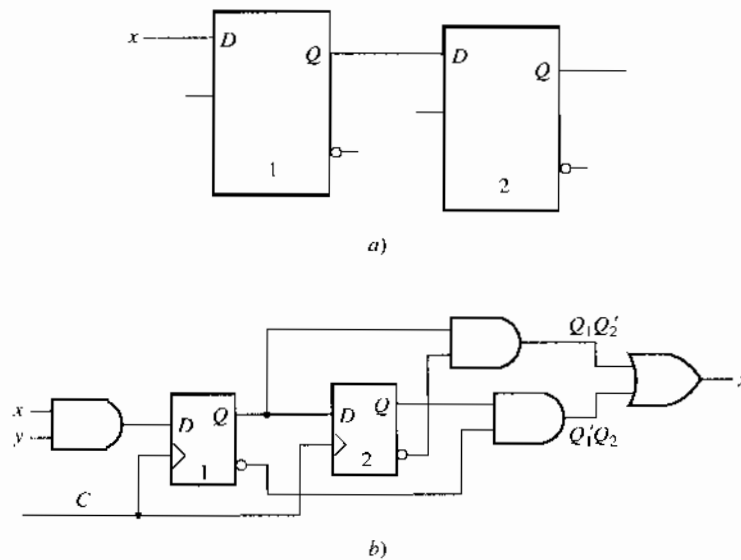


Figura 21. Circuito de Moore que realiza el ejemplo 8.

Puesto que la salida dependerá únicamente del estado, la paridad se determina a partir de la memoria de los últimos 2 bits de entrada x . Esto sugiere dos flip-flops, digamos flip-flops D . Un registro consistente en dos flip-flops D , denominados 1 y 2, se muestra en la figura 21a.¹² Advertiéndose que, en un tic tac de reloj determinado, Q_1 tendrá cualquier valor x que tuvo en el pulso de reloj precedente. De manera similar, Q_2 tendrá cualquier valor Q_1 que tuvo en el pulso de reloj anterior. Utilizando esta información, construya mapas lógicos de tres variables (x , Q_1 y Q_2) para Q_1^+ y Q_2^+ y confirme que $Q_1^+ = x$ y $Q_2^+ = Q_1$. Puesto que la salida depende sólo del estado, un mapa lógico para la salida será precisamente uno de dos variables (Q_1 y Q_2). Dibuje este mapa y determine una expresión para la salida.

Respuesta¹³

El requisito de que el proceso continuará sólo cuando $y = 1$ puede satisfacerse mediante una compuerta AND con entradas x y y . La entrada al primer flip-flop seguirá siendo 0 cuando $y = 0$.

Aunque será x cuando $y = 1$. El circuito completo, incluso el decodificador de salida, se muestra en la figura 21b. (Compruebe.)

Lo que se hizo en este ejemplo parece un poco engañoso: no se dibujó un diagrama de estados, no se crearon tablas de estados, no se hicieron asignaciones, etc. Hemos visto que dos flip-flops permiten la realización de la función y que, en consecuencia, existe un máximo de cuatro estados. Como ejercicio, construya una tabla de estados utilizando las asignaciones (00, 01, 10 y 11) como nombres de estados y construya una columna separada para la salida, independiente de la entrada. Utilizando esta tabla, confirme los mapas lógicos que construyó previamente.

4 EQUIVALENCIA DE ESTADOS Y MINIMIZACIÓN DE MÁQUINA

El ejercicio 1 en la sección 1 (y también el ejemplo 7) demostró que es posible construir más de un diagrama de estados (o tabla de estados) para satisfacer la descripción de un problema de

¹² Los registros se presentaron en el capítulo 5, repase si es necesario.

¹³ $z = Q_1Q_2' + Q_1'Q_2 = Q_1 \oplus Q_2$.

diseño. Si una tabla de este tipo tiene nueve estados y otra cuenta con cuatro, la primera requiere dos veces más flip-flops en su implementación que la segunda. Pero no se necesita almacenar más información en la primera máquina que en la segunda. Debe suceder que sin importar cuál sea la tarea efectuada por alguno de los nueve estados, también la efectúan otros estados, originando que alguno de los nueve estados sea superfluo.

Sería de gran beneficio detectar y eliminar estos estados superfluos, dejando de esa manera una tabla de estados reducida. Ya hemos mencionado la posible reducción en la complejidad del circuito. Una vez que se implementa una máquina reducida, lo que es incluso posiblemente de más valor es que la complejidad reducida hace que la verificación (la determinación experimental de fallas en una máquina) resulte considerablemente más simple. El propósito de esta sección es formular procedimientos para reducir una tabla de estados determinada en una que lleve a cabo la misma función con menor número de estados.

Distinguibilidad y equivalencia

Una máquina de estados finita opera recibiendo una secuencia de símbolos de entrada, efectuando transiciones de estado y emitiendo símbolos de salida. La secuencia de entrada viene a transformarse en una secuencia de salida. Volvamos a la tabla de estados de la figura 11 y supongamos que el circuito está en el estado B cuando ocurre una entrada 1. Se efectúa una transición al estado D. Describimos lo anterior afirmando que el estado D *sucede* al estado B bajo una entrada 1, o que D es el *sucesor* con 1 de B. Suponga ahora que llegó la secuencia de entrada más larga 011 cuando la máquina está inicialmente en el estado A. El estado final que se alcanza será E. (Trace la secuencia de estados que se encuentran y verifique.) De modo que E podría denominarse el sucesor con 011 de A. En general,

Si se aplica una secuencia de entrada X a una máquina de estados finitos que está en el estado S_i y la máquina efectúa una transición final al estado S_j , entonces S_j es el sucesor con X de S_i .

Si la misma secuencia de entrada X se aplica dos veces a una máquina, la primera cuando la máquina se encuentra en el estado S_i y la segunda cuando está en el estado S_j , las dos secuencias de salida producidas quizás sean o no las mismas. Si son las mismas, no seremos capaces de distinguir los dos estados iniciales por medio de esa secuencia de entrada particular. Suponga ahora que la secuencia de salida es siempre la misma, iniciando a partir de cada uno de los dos estados, sin importar qué secuencia de entrada se aplique. Claramente, los dos estados nunca podrían distinguirse uno del otro. Lo anterior nos lleva a la siguiente definición:

Dos estados S_i y S_j en una máquina de estados finitos son equivalentes si se produce la misma secuencia de salida en respuesta a una secuencia de entrada, empezando en cualquiera de los estados, y si esto es cierto para toda secuencia de entrada finita.

Si se fuera a verificar la equivalencia potencial de dos estados utilizando esta definición, sería necesario un largo tiempo para verificar todas las secuencias de entrada posibles. Es evidente que se necesita una prueba más corta. Suponga que, cuando una máquina se inicia en dos estados diferentes, las secuencias de salida producidas por la misma secuencia de entrada *no* son las mismas. En ese caso es posible distinguir los dos estados. Formulamos el siguiente enunciado:

Dos estados S_i y S_j de una máquina son distinguibles si y sólo si existe al menos una secuencia de entrada finita que produce diferentes secuencias de salida empezando primero desde el estado S_i y luego desde el S_j . Si la secuencia distinguible tiene longitud k , entonces se dice que los dos estados son distinguibles en k .

Minimiz

PS	NS,z		PS	NS,z	
	x = 0	x = 1		x = 0	x = 1
A	B,0	D,1	A	B,0	D,1
B	C,1	D,1	B	C,1	D,1
C	C,0	A,0	C	E,0	A,1
D	A,0	C,1	D	A,0	C,1
	a)		E	B,0	D,1
				b)	

Figura 22. Máquinas de ejemplo.

Como ejemplo, considere los estados A y D en la tabla de estados dada en la figura 22a como estados iniciales. Las salidas son las mismas para cualquier entrada de longitud 1 (0 o 1). En consecuencia, los estados A y D no son distinguibles en 1. Considérese ahora una entrada 00, de longitud 2, empezando a partir de A, la salida es 01, aunque al empezar desde D, ésta es 00. Por consiguiente, los estados A y D son distinguibles en 2. Esto nos lleva a la siguiente definición:

Dos estados que no son distinguibles en k son equivalentes en k.

La definición previa para la equivalencia puede expresarse ahora como sigue:

Dos estados de una máquina son equivalentes si son equivalentes en k para toda k.

En la discusión anterior nos hemos concentrado en las secuencias de salida en respuesta a una secuencia de entrada. No se ha puesto ninguna atención a los siguientes estados que resultan a lo largo del proceso. En la figura 22a, por ejemplo, encontramos que los estados A y D son equivalentes 1; en ese caso probamos una secuencia de entrada de longitud 2. Suponga, en vez de eso, que consideramos los siguientes estados después de la primera entrada. Los siguientes estados no son los mismos empezando desde los estados A y D. Además, es evidente que, utilizando aquellos estados siguientes como estados presentes, no resulta la misma salida para cada entrada. Por tanto, los estados originales A y D son distinguibles.

Continuaremos con esta línea, utilizando la figura 22b, la cual es casi la misma tabla que la de la figura 22a. En este caso los estados A, D y E son todos equivalentes en 1. Además, los siguientes estados a partir de A y E son los mismos para cada entrada. Así, luego del primer bit de entrada, la transición a partir de cada uno de los estados A y E será hacia el mismo estado siguiente; las salidas a partir de ahí serán exactamente iguales. En consecuencia, A y E son equivalentes.

Minimización de máquina

La discusión anterior proporciona una clave en cuanto a la manera de que se determinan los estados de una máquina que son equivalentes entre sí. Empezando a partir de una secuencia de entrada de longitud 1, agrupamos aquellos estados que son equivalentes en 1. Estos estados son distinguibles de los otros. Luego examinamos los siguientes estados para decidir su distinguibilidad, etc. Los detalles de este proceso se describen mejor con un ejemplo.

EJEMPLO 9

En la figura 23a se presenta una tabla de estados. El objetivo es determinar todos los grupos de estados equivalentes y reducir la tabla a una que tenga el número mínimo de estados.

PS	NS, _z			PS	NS, _z	
	x = 0	x = 1			x = 0	x = 1
A	D,1	G,1	$P_1 = \{ADF; BCEG\}$	A → S ₁	S _{3,1}	S _{4,1}
B	C,0	D,1		BCE → S ₂	S _{2,0}	S _{3,1}
C	E,0	F,1	$P_2 = \{ADF; BCE; G\}$	DF → S ₃	S _{3,1}	S _{2,1}
D	F,1	B,1		G → S ₄	S _{1,0}	S _{3,1}
E	B,0	F,1	$P_3 = \{A; DF; BCE; G\}$			
F	D,1	C,1				
G	A,0	D,1	$P_4 = P_3$			

a)

b)

c)

Figura 23. División y minimización de máquina.

De acuerdo con el plan, empezamos identificando aquellos estados que son distinguibles con una secuencia de entrada de longitud 1. Encontramos que el grupo de estados A, D, F tiene la misma salida para $x = 0$; también tienen la misma salida para $x = 1$. Por consiguiente, no son distinguibles con una secuencia de entrada de longitud 1. De manera similar, confirme que los estados B, C, E, G son indistinguibles con una secuencia de entrada de longitud 1. Sin embargo, estos dos grupos de estados son distinguibles entre sí. Así, la totalidad de los estados en la tabla puede *dividirse* en dos bloques de estados, escritos como sigue: $P_1 = \{ADF; BCEG\}$. Dentro de cada bloque, los estados son indistinguibles con una secuencia de entrada de longitud 1, pero aquellos en un bloque son distinguibles de aquellos en el otro.

Examinaremos ahora los estados sucesores de todos los estados en cada bloque, uno a la vez. Si, para cada símbolo de entrada, los siguientes estados de todos los estados en un bloque no se encuentran en el mismo bloque sino que caen en dos distintos, entonces los dos subbloques son distinguibles. En consecuencia el bloque original debe subdividirse. De tal manera, para $x = 1$, los siguientes estados del bloque BCEG son DFFD; todos éstos se encuentran en el bloque ADF. Sin embargo, para $x = 0$, los siguientes estados a partir del bloque BCEG son CFBA; todos excepto el siguiente estado a partir del estado G están en el mismo bloque. Por tanto, el bloque BCEG debe subdividirse en dos bloques, BCE y G. La división que resulta es $P_2 = \{ADF; BCE; G\}$, como se muestra en la figura 23b; ésta es una versión *refinada* de P_1 .

Se encontró que los siguientes estados a partir de cualquier bloque en la división P_2 estaban en el mismo bloque; por tanto, estos estados siguientes tendrán las mismas salidas para cada bit de entrada. (Esto es debido a que las salidas fueron las mismas, incluso para los bloques más grandes en la división P_1 .) Estos nuevos estados en cada bloque son, entonces, equivalentes 1. En consecuencia, sus estados predecesores son equivalentes en 2.

El proceso se repite ahora con la división P_2 . También en este caso consideramos cada bloque uno a la vez y, para cada entrada, examinamos sus siguientes estados para observar si todos ellos caen en el mismo bloque nuevo. (Claramente, no es necesario examinar los bloques que contienen un solo estado.) Para cada entrada, los siguientes estados a partir del bloque BCE caen en el mismo bloque. Esto también es cierto para $x = 0$ para el bloque ADF; sin embargo, para $x = 1$, los estados siguientes para el bloque ADF son GBC. Estos estados siguientes no se encuentran en el mismo bloque, por lo que se necesita un refinamiento adicional de P_2 , como se indica en la figura 23b. Los estados en cada nuevo bloque son equivalentes en 3.

El proceso debe repetirse en los bloques multiestado en la división P_3 . Efectúe el proceso y confirme que no se necesitan refinamientos adicionales de la división. Los estados dentro de cada bloque no pueden distinguirse; entonces, son equivalentes. Esta división final recibe el nombre de *división de equivalencia*. Cada uno de los cuatro bloques en la división equivalente constituye un estado de la máquina a la cual cada uno de los estados originales en ese bloque es equivalente. Si estos estados se indican por medio de S_i , es posible construir una tabla de estados reducida, como se muestra en la figura 23c. ■

La descripción de este proceso es bastante más larga que el esfuerzo real implicado en llevarlo a cabo.¹⁴ Advierta, en este ejemplo, que la máquina reducida sólo necesita dos flip-flops en su implementación, en tanto que la tabla original requiere tres.

El tema de esta sección constituye el segundo paso en el proceso de diseño general que se describió en la sección anterior. Como se señaló antes, cuando se construye inicialmente una tabla de estados para satisfacer las especificaciones de un problema de diseño, no es necesario dedicar demasiado tiempo para asegurar que no hay estados redundantes. Cualesquiera estados redundantes que se introducen con anterioridad siempre pueden eliminarse en el paso de minimización de la máquina.¹⁵

5 MÁQUINAS CON RANGOS DE MEMORIA FINITA¹⁶

La memoria es lo que distingue a los circuitos secuenciales de los combinatorios. La información almacenada en la memoria, junto con una secuencia de entrada, determina una secuencia de salida. ¿Pero cuántos datos pasados son necesarios para que la máquina recuerde? ¿Es necesario para la máquina recordar *únicamente* entradas pasadas, o es posible que su comportamiento futuro sea determinado a partir de la entrada presente y de una memoria de *salidas* pasadas? ¿O es necesaria la memoria tanto de entradas *como* de salidas?

Examinaremos tres clases de máquinas en esta sección. En los tres casos respectivos, la salida presente se determina por medio de la entrada presente más:

1. Un número limitado de entradas inmediatamente precedentes.
2. Un número limitado de salidas inmediatamente precedentes.
3. Un número limitado de entradas y salidas inmediatamente precedentes.

Se afirma que todas las máquinas tienen *rangos de memoria finita*.

No todas las máquinas de estados finitos tienen esta característica; no la tiene, por ejemplo, una máquina con un estado en espera. No puede producir una salida mientras está en este estado, no importa si permanece mucho tiempo ahí, incluso si recibe una secuencia de entrada aceptable. Cada una de estas tres clases de máquina de memoria finita se explicará y se implementará en ciertas estructuras específicas.

Máquinas con memoria de entrada finita

Una posible definición formal de las máquinas que se expondrá en esta sección es la siguiente.

Una máquina de estados finitos M se dice que tiene memoria de entrada finita de rango (u orden) de memoria m si el estado presente de M puede determinarse exclusivamente a partir de los m símbolos de entrada precedentes pero no de un número menor que m .

Es claro a partir de la definición que un circuito que implementa la memoria es un registro de corrimiento de m flip-flops en el cual se almacenan las últimas m entradas. El estado presente consiste en las salidas de los m flip-flops en el registro. Una implementación *canónica* consta de un registro de corrimiento de m flip-flops y de un decodificador de salida combinatorio, como se muestra en la figura 24.

¹⁴ El proceso es de naturaleza algorítmica. Eso quiere decir que es posible producir software para efectuarlo. Aquí nos concentraremos en los principios.

¹⁵ El tratamiento aquí se ha limitado a máquinas completamente especificadas. Las máquinas especificadas de manera incompleta requieren la introducción de varios conceptos adicionales que necesitan una amplia formulación. Estos conceptos se aplican también a máquinas asíncronas, y se tratarán en el capítulo 7. Si se salta el capítulo 7, hará lo mismo con la cobertura de las máquinas síncronas incompletamente especificadas.

¹⁶ Esta sección puede omitirse sin que se afecte la preparación del material que sigue.

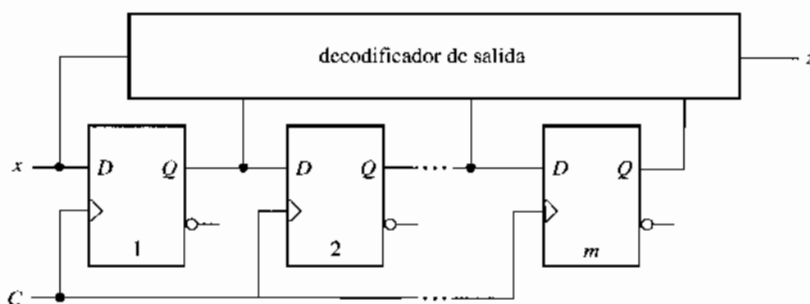


Figura 24. Implementación canónica de una máquina de memoria de entrada finita.

El registro de corrimiento en la figura 24 es un convertidor serie-paralelo. La información de entrada llega de manera secuencial y se almacena en el registro de corrimiento. Cuando llega el último bit de una secuencia de entrada de longitud apropiada, se aplican tanto esa entrada como la información almacenada previamente a la lógica combinatoria al mismo tiempo, en paralelo; la lógica del decodificador produce entonces la salida deseada.

Puesto que se conoce el estado de una máquina de memoria de entrada finita de rango m después de una secuencia de entrada de m bits, la salida se volverá conocida cuando arribe el siguiente bit. Por consiguiente, una máquina de este tipo también puede definirse como aquella cuya salida se determina por medio de la entrada presente y la secuencia de entrada precedente de m bits.

Es posible simplificar el procedimiento de diseño de una máquina con rango de memoria finito si las especificaciones del problema de diseño nos permiten reconocer su naturaleza; únicamente necesita diseñarse el decodificador de salida.

EJEMPLO 10

Un circuito secuencial síncrono con una sola línea de entrada x y una sola línea de salida z se diseñará de manera que produzca una salida $z = 1$ cada vez que un símbolo de entrada complete una secuencia de 4 bits de entrada idénticos; en otro caso, la salida será 0.

Justo antes de la recepción de cada símbolo de entrada, la máquina debe recordar únicamente los 3 bits precedentes. Se produce entonces un 1 o un 0 con base en aquellos 3 bits y en el bit de entrada presente. Ésta, por tanto, es una máquina que tiene memoria de entrada finita de rango 3. Indicando los estados Q con los subíndices 1, 2 y 3, de izquierda a derecha, los minitérminos son 0000 y 1111. Por consiguiente, la salida se escribe con facilidad como

$$z = xQ_1Q_2Q_3 + x'Q_1'Q_2'Q_3'$$

La implementación canónica del circuito se muestra en la figura 25. ■

Aunque el ejemplo anterior incluye sólo una línea de entrada, el concepto y la implementación canónica se aplican a cualquier número de líneas de entrada. En cada una de éstas se necesita un registro de corrimiento independiente.

Además de las entradas principales, algunas máquinas tienen una o más entradas de control para cambiar las instrucciones correspondientes a la generación de una salida. El valor presente de aquellas entradas es lo que efectúa el control; no es necesario almacenar las entradas de control pasadas. Como caso ilustrativo, en el ejemplo precedente podría haber una entrada de control x_c que, de acuerdo con las condiciones del problema dadas anteriormente, permitiera que la salida

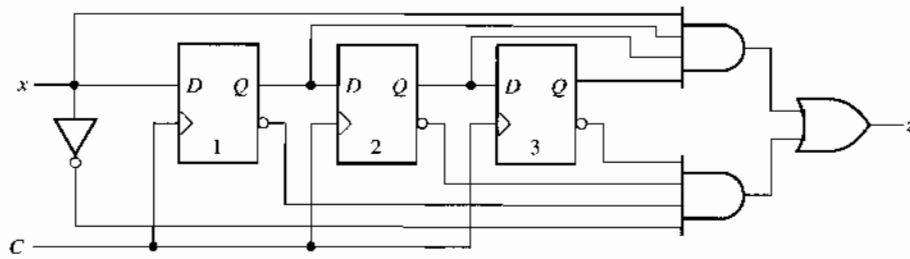


Figura 25. Implementación canónica del ejemplo 10.

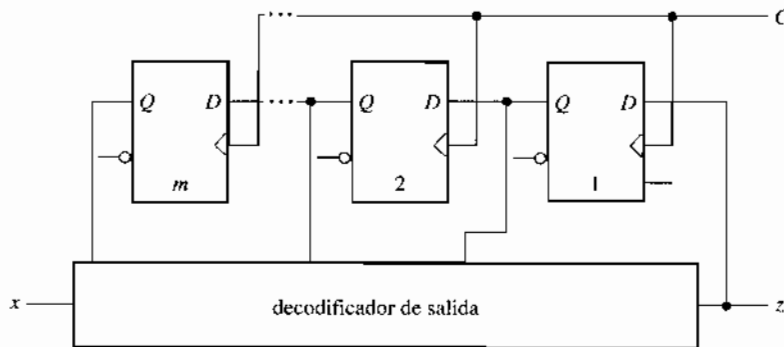


Figura 26. Forma canónica de la máquina de memoria de salida finita.

se volviera 1 sólo si $x_c = 1$. Para $x_c = 0$ la salida podría especificarse como algo más en términos de la entrada principal x y sus valores pasados. En la implementación de la máquina, el registro de corrimiento de la figura 25 no cambiaría; sólo se modificaría la lógica del decodificador de salida.

Máquinas con memoria de salida finita

En la segunda clase de máquinas que se está considerando, las *salidas* precedentes son las que deben recordarse y no las entradas. La definición de esta clase de máquinas es como sigue:

Se dice que una máquina secuencial M tiene memoria de salida finita de rango (u orden) de memoria m si la salida presente de M puede determinarse a partir de la entrada presente y los m (pero no menos) símbolos de salida inmediatamente precedentes.

También en este caso la definición plantea con claridad que es posible implementar la memoria con un registro de corrimiento (registro izquierdo) de m flip-flops en el cual se almacenan las m salidas precedentes. Entonces, cuando llega el siguiente símbolo de entrada, la salida se determina tanto por esta entrada como por las salidas almacenadas con anterioridad. La implementación canónica se presenta en la figura 26. La entrada al registro de corrimiento es el símbolo de salida más reciente.

También resulta claro que, si hay más de una línea de salida, la implementación canónica requerirá más de un registro de corrimiento.

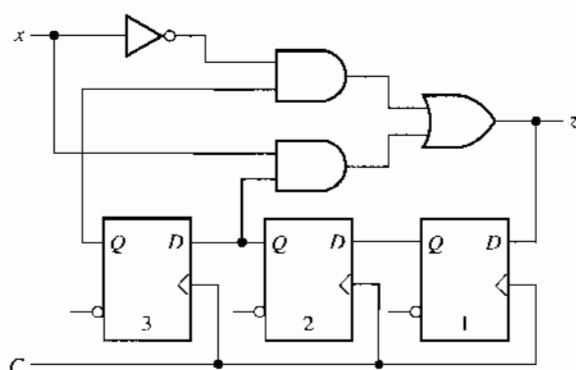


Figura 27. Máquina de memoria de salida finita que implementa al ejemplo 11.

EJEMPLO 11

Diseñaremos una máquina secuencial de una entrada, una salida. Para una entrada $x = 1$, la salida es igual a la salida que se produjo dos periodos de reloj anteriores; y para $x = 0$, la salida debe ser igual a la que se produjo tres periodos de reloj anteriores.

Lo más que la máquina debe recordar son los tres símbolos de salida precedentes. Por consiguiente, la máquina que se desea tiene una memoria de salida finita de rango 3. Una expresión para la salida se escribe sin dificultades. Si denominamos los estados Q_1 a Q_3 , entonces cuando $x = 1$, la salida debe ser Q_2 ; y cuando $x = 0$, (lo que equivale a $x' = 1$), la salida debe ser Q_3 . Por tanto,

$$z = xQ_2 + x'Q_3$$

(Verifique el valor de z cuando $x = 1$ y cuando $x = 0$.) La implementación se muestra en la figura 27. ■

Ejercicio 8. A partir del enunciado del problema del ejemplo 11, dibuje un mapa lógico para la salida, con la entrada y los tres estados como variables del mapa. A partir de este último, escriba una expresión mínima de suma de productos y confirme la expresión en el ejemplo. (Esta implementación no depende del reconocimiento de la naturaleza del circuito como una memoria de salida finita.) ♦

Máquinas de memoria finita

La tercera clase de máquinas de estado finito que se explica depende no sólo de un número fijo de entradas pasadas o de un número fijo de salidas pasadas, sino de ambos. La definición formal es la siguiente:

Una máquina secuencial M es una máquina de memoria finita de rango m si es posible determinar el estado presente exclusivamente a partir de los m_i (pero no menos) símbolos de entrada precedentes y los m_o (pero no menos) símbolos de salida precedentes; el rango es $m = \max \{m_i, m_o\}$.

Se podría conjeturar que una implementación canónica de esta máquina surgiría de las dos implementaciones canónicas de las figuras 24 y 26 incluyendo dos registros de corrimiento, uno para almacenar las m_i entradas pasadas y otro para almacenar las m_o salidas pasadas. Si bien una implementación de este tipo es posible en algunos casos, resulta que no es posible universalmente.

Considere una máquina de memoria de entrada finita. De la definición que acaba de darse, esta máquina es también una máquina de memoria finita. Esto es, si el estado presente se determina exclusivamente mediante los primeros m_i símbolos de entrada, conocer también los primeros m_o símbolos de salida no desmerecerá lo anterior. Esta máquina de memoria finita, sin embargo, no es una de memoria de salida finita; es de memoria finita en virtud de que tiene memoria de entrada finita. Es posible plantear un argumento similar en cuanto a que una máquina con memoria de salida finita es, en virtud de este hecho, una máquina de memoria finita, pero no es una de memoria de entrada finita.

Lo inverso no es cierto. Es decir, es posible que una máquina sea de memoria finita sin que tenga memoria de entrada finita ni memoria de salida finita. Establecer la validez de esta afirmación requiere introducir varios conceptos y algoritmos adicionales que nos extenderían demasiado. Por tanto, abandonamos una consideración adicional del tema en esta parte, aunque ofreceremos algunos problemas al final del capítulo de manera que el lector pueda explorarlo hasta cierto punto.

6 CONTADORES SÍNCRONOS

Consideramos ahora una clase de máquinas secuenciales que efectúa un tipo particular de operación. Un *contador* es una máquina secuencial que, empezando en un estado particular, efectúa ciclos a través de una secuencia fija de estados y luego regresa a su estado inicial; de ahí en adelante, repite este proceso. El número de estados distintos en el contador se conoce como su *número de módulo*.

En algunos casos, la información útil del contador quizá sea simplemente el estado en el cual se encuentra. En este caso, no hay circuito decodificador de salida ni otras líneas de salida sino sus salidas de flip-flop. En otras situaciones, es posible que se requiera una salida en lugar de un estado. En los contadores síncronos, la señal que excita al contador muy a menudo es el reloj. En otras ocasiones, también se proporcionan otras entradas (denominadas *de control*). (Igualmente es posible que los contadores sean síncronos; este tipo se considerará en el capítulo 7.)

Los contadores se usan para varios fines. Uno común es extender la escala de tiempo, esto es, introducir retardo en la inevitable marcha del reloj. Lo anterior se efectúa produciendo una señal de salida (o control) para cada k periodos de reloj; esta señal, más que el reloj, controla entonces la temporización de una operación subsecuente. Otro fin del contador corresponde a producir palabras secuenciales en algún código específico. Desde luego, tan sólo el conteo sencillo es un propósito importante, como por ejemplo, contar cuántas veces se ha efectuado algún proceso. Éste resultaría muy útil si

- El contador fuera borrado (se restableciera en 0) cuando se pone en marcha o después de realizar su conteo, o
- El contador se fijará en algún valor específico.

Esto se lleva a cabo por medio de las entradas BORRAR y RESTABLECER.

Contadores de modo simple

Se dice que un contador será de *modo simple* si la única entrada externa es el reloj y las únicas salidas son los estados —las salidas de flip-flop. El contador se describe especificando

- Su número de módulo
- El código asignado a los estados

El número de flip-flops necesarios en un contador está implícito en su número de módulo. De tal manera, un contador de módulo k requerirá $\lceil \log_2 k \rceil$ flip-flops. Los contadores de módulo 6 o

Distancia				Distancia			
Binario	Unitaria	Progresivo	Uno-caliente	Binario	Unitaria	Progresivo	Uno-caliente
000	000	000	00000	000	000	0000	0000000
001	001	100	10000	001	001	1000	1000000
010	011	110	01000	010	011	1100	0100000
011	010	111	00100	011	010	1110	0010000
100	110	011	00010	100	110	1111	0001000
101	100	001	00001	101	111	0111	0000100
				110	101	0011	0000010
				111	100	0001	0000001

a)

b)

Figura 28. Códigos que se usan en los contadores de módulo 6 y módulo 8. a) Módulo 6. b) Módulo 8.

módulo 8, contando en código binario o Gray, contarán con tres flip-flops; en terminología común los denominados "contadores de 3 bits".

Si se especifica el código en el cual ocurre el conteo, no hay motivo para dar a los estados nombres arbitrarios (como letras del alfabeto) y efectuar después una asignación de estado, más bien, se da a cada estado sucesivo una asignación con base en el código que se está utilizando. Resulta conveniente asignar al estado inicial del contador la palabra de código 00...0, a menos que exista alguna razón para no hacerlo. (Regrese al capítulo 1 para repasar el tema de códigos.) Como un hecho, el problema de la asignación de estados, tan importante en las máquinas consideradas hasta ahora, desaparece en un contador. Los códigos que representan los estados se especifican de antemano.

El número de palabras de código se fija por medio del número de módulo. En la figura 28 se muestran varios códigos correspondientes a los números de módulos 6 y 8. Empezando en la palabra de código 00...0, el contador efectuará un ciclo a lo largo de cada palabra de código y regresará a 00...0 después de la última palabra en cada código.

Contadores de distancia unitaria

Después que se ha especificado el número de módulo de un contador, la siguiente pregunta que surge es, ¿qué código debe usarse al realizar las sucesivas asignaciones de estado? La respuesta más simple quizás sea: el binario. La desventaja de este código es que más de un valor de bit cambia en un periodo de reloj. Así, al pasar de 001 a 010, tanto el segundo como el tercer bit (contando de izquierda a derecha) deben cambiar de valor. Esto equivale a que más de una salida de flip-flop deba cambiar en forma simultánea. Si es necesario que un cambio ocurra incluso un poco antes que el (los) otro(s), es posible que haya una transición momentánea hacia el estado equivocado, o incluso hacia un estado inválido, que no se encuentra entre los estados del contador. Por esta razón, resulta preferible un código cuya *distancia* sea 1.¹⁷

Los códigos en la segunda y tercera columnas de la figura 28a son de distancia unitaria. Suponga que se elige el código en la segunda columna. Puesto que no hay entradas (además del reloj) ni salidas, no habrá información de entrada/salida en el diagrama de estados. (Dibuje usted este último.) Puesto que no hay decodificador de salida, el único hardware en el circuito además de los tres flip-flops es el decodificador de estado. Como los estados se identifican por medio de

¹⁷ La distancia entre dos palabras de código es el número de bits que debe complementarse en una palabra para transformarla en otra palabra. Un código en el cual la distancia entre cada par de palabras de código consecutivas es k es un código de *distancia k*. Véase el capítulo 1 para una explicación de códigos.

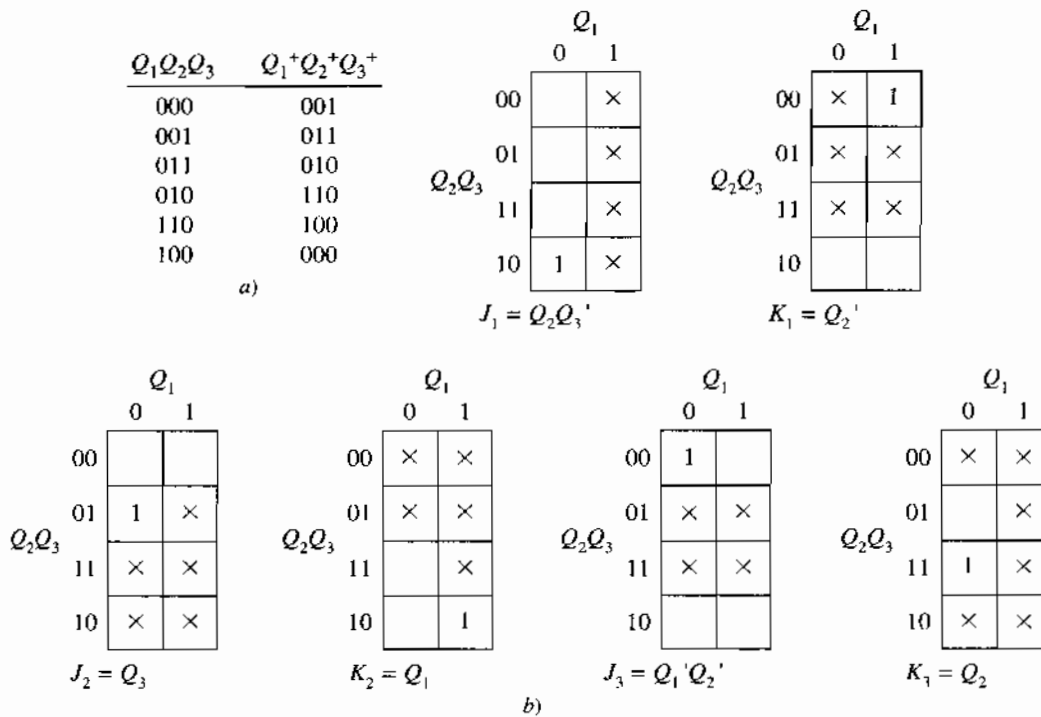


Figura 29. Diseño de contador de módulo 6 de tres bits.

su asignación de acuerdo con el código, la tabla de estados resultantes es la tabla de transición que se muestra en la figura 29a. Suponiendo flip-flops JK, utilizamos las tablas de requerimientos de excitación de la figura 17 del capítulo 5 para construir los mapas lógicos para las excitaciones J y K . Esto se hace línea por línea en la tabla de excitación para cada columna. (Estas tablas aparecen en la cubierta interior del libro. Podría sacarle una copia y tenerla a mano para que pueda consultarla sin tener que buscar afanosamente en el libro cada vez que desee recurrir a las tablas.)

A partir de los requerimientos de excitación en la figura 17, capítulo 5, $J = 1$ únicamente para la transición de 0 a 1. En la tabla de transición de este ejemplo (para el flip-flop 1), esto ocurre sólo cuando $Q_1Q_2Q_3 = 010$. Por consiguiente, se anota un 1 en la celda correspondiente en el mapa lógico correspondiente a J_1 en la figura 29b. Además, J_1 es un valor no relevante para el estado presente $Q_1 = 1$, independientemente de los otros estados. Asimismo, todas las entradas J y las K son valores irrelevantes para los estados presentes que nunca ocurren (110, 111). Todo lo anterior confirma el mapa lógico para J_1 en la figura 29b.

Ejercicio 9. Use el mismo método para construir los mapas lógicos relativos a las otras entradas J y K . Confirme sus resultados utilizando la figura 29b. ♦

Ejercicio 10. A partir de los mapas de la figura 29b, construya el hardware combinatorio del decodificador de estado. Verifique su circuito con la implementación de la figura 30. ♦

Contadores de anillo

Otro código de distancia unitaria en la figura 28a es el progresivo. Un contador designado para contar en este código recibe el nombre de *contador de anillo*. La tabla de estados, que es tam-

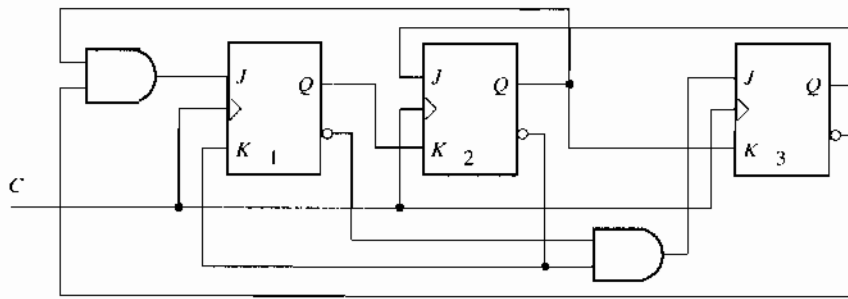


Figura 30. Implementación del flip-flop JK del contador de módulo 6.

		Q_1	
		0	1
$Q_1Q_2Q_3$	$Q_1+Q_2+Q_3+$	00	1
		01	×
		11	
		10	×
			1

$D_1 = Q_3'$

		Q_1	
		0	1
Q_2Q_3	00		1
	01		×
	11		1
	10	×	1

$D_1 = Q_3'$

		Q_1	
		0	1
Q_2Q_3	00		
	01		×
	11	1	1
	10	×	1

$D_3 = Q_2$

a)
b)

a)

b)

Figura 31. Tabla de transición y mapas de excitación para el contador de anillo trenzado.

bién la tabla de transición (puesto que los estados se identifican mediante sus asignaciones), se ilustra en la figura 31a. Esta vez, vamos a suponer que se utilizarán flip-flops D . Entonces las excitaciones son los estados siguientes.

Ejercicio 11. Dibuje mapas lógicos para las entradas D a partir de la tabla de transición. Realice la verificación utilizando la figura 31b, pero sólo *después* de haber dibujado los suyos propios.

Es claro que la salida de un flip-flop es la excitación para el siguiente, salvo que la salida del último flip-flop (la cola) se complementa antes de convertirse en la entrada para el primer flip-flop. (Por esta razón se le asigna el nombre de *contador de anillo trenzado* o alguna variación folclórica de esto.) En consecuencia, este contador no es sino un registro de corrimiento en serie con su salida complementada que se alimenta a su entrada. Como práctica, dibuje la implementación del circuito.

Estados indeterminados

Los contadores de anillo tienen un problema que no es evidente a partir de la figura 31. ¿Qué sucedería si, por alguna razón, el contador entrara a uno de los estados sin uso (101, o 010?; véase la figura 28a). Esto podría ocurrir cuando la energía eléctrica se activara por primera vez, por ejemplo, o como consecuencia del ruido en el circuito. Nada importante pasaría si el siguiente estado no fuera uno sin uso. En este caso, el *contador* reanudaría correctamente a partir del siguiente pulso de reloj. Pero, por otro lado, si cada estado siguiente para los pulsos de reloj sub-

Contad

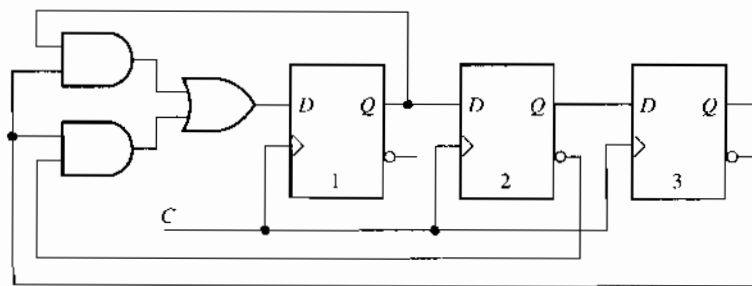


Figura 32. Contador de anillo de módulo 6 con autocorrección.

secuentes es un estado sin uso, el contador *fallará*. Un estado en el cual el contador falla de manera que el conteo no puede proseguir recibe el nombre de *estado indeterminado*.

Ahora bien, ¿cuál es la situación en la figura 31? Si sucede que el estado presente corresponderá al sin uso $Q_1Q_2Q_3 = 010$, entonces el estado siguiente será $Q_1^+Q_2^+Q_3^+ = D_1D_2D_3 = Q_3'Q_1Q_2 = 0'01 = 101$; éste es el otro estado sin uso. (Siguiendo un proceso similar, demuestre que el siguiente estado después de éste será otra vez 010.) Luego de que el circuito entra a uno de los dos estados sin uso, continúa realizando ciclos entre ellos; nunca volverá a la secuencia de conteo. Por consiguiente este contador sería defectuoso e inútil.

Puesto que el problema lo provocan las ecuaciones de excitación, originando una secuencia entre estados sin uso, es posible resolverlo rehaciendo las ecuaciones de excitación. No todas las ecuaciones necesitan modificarse. Si quisiéramos mantener el uso del registro de corrimiento, deberíamos concentrarnos en la ecuación de excitación de sólo el primer flip-flop. Considerando el mapa para D_1 , el problema de indeterminación surge debido a que definimos como un 1 el valor irrelevante en la posición 010 para formar un cubo de orden 2. En vez de eso, vamos a reasignar el valor y considerarlo como un 0. En ese caso, la expresión para D_1 se convierte en

$$D_1 = Q_2'Q_3' + Q_1Q_3'$$

Para el estado presente sin uso $Q_1Q_2Q_3 = 010$, el siguiente estado de Q_1 será $Q_1^+ = D_1 = 0$, y de ese modo el siguiente estado corresponderá a 001. ¡Hemos salido del estado indeterminado!

Ejercicio 12. Suponga que el estado presente es el otro estado sin uso, 101. Recurriendo a la expresión para D_1 y a las anteriores para D_2 y D_3 , determine el estado siguiente y explique si se ha salido del estado indeterminado. ♦

El diseño del contador de anillo incorporando el cambio precedente se ilustra en la figura 32. Es un diseño *autocorregido* en el que se han evitado los estados indeterminados. El costo en el peor de los casos de este diseño es un retardo de dos pulsos de reloj en el conteo si el contador incorpora de modo inadvertido uno de los dos estados indeterminados. Si bien se han presentado flip-flops individuales, resulta apropiado un registro de corrimiento MSI para la implementación.

Contadores multimodo

Un contador se denomina *multimodo* si cuenta, además del reloj, con entradas externas y, posiblemente, salidas externas además de las salidas de estado. Es "multimodo" debido a que la secuencia de conteo podría depender no sólo del reloj sino también de algunas otras señales de control. Igualmente, quizás se proporcionen líneas de salida especiales además de las salidas de flip-flop.

Un contador de estas características podría emplearse, por ejemplo, en un sistema en el cual se efectuarán varias operaciones consecutivas. Sólo cuando una operación se completa se inicia

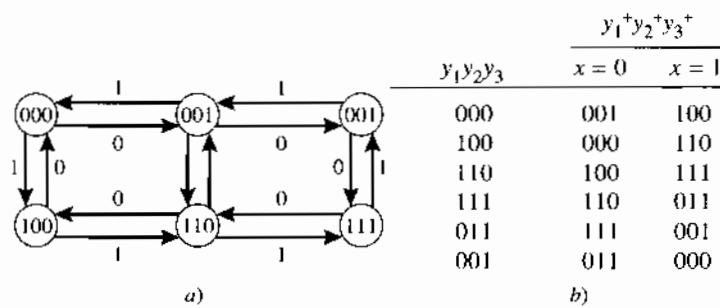


Figura 33. Tabla de estados y tabla de transición para un contador ascendente-descendente de módulo 6.

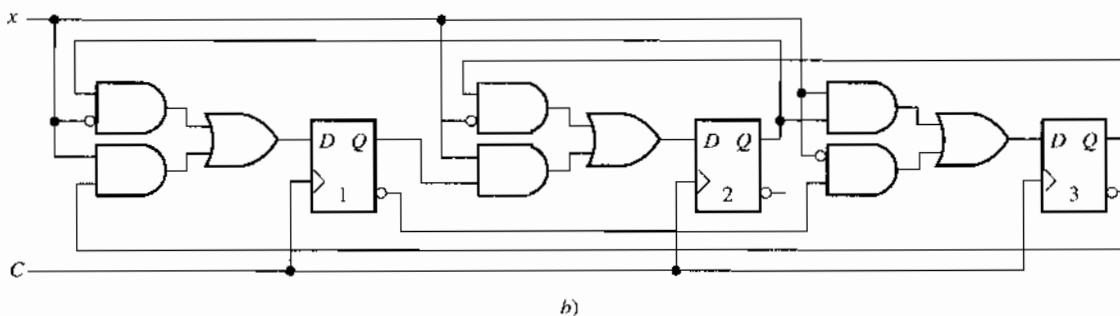
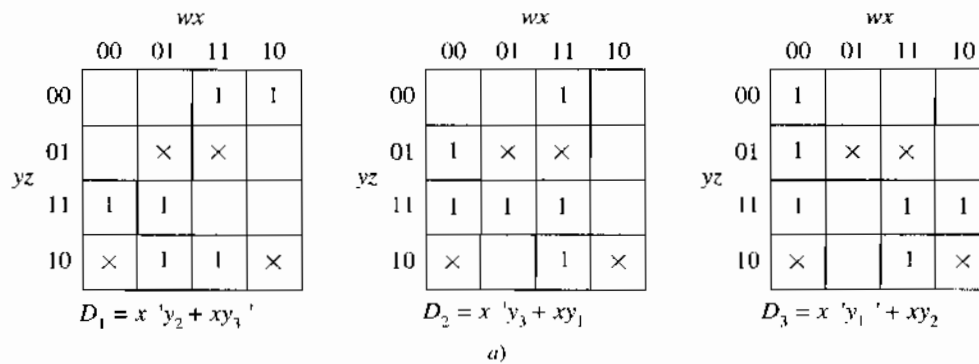


Figura 34. Diseño de un contador ascendente-descendente de módulo 6.

la siguiente. De ese modo una señal de control que indique el fin de una operación particular incrementará el conteo, provocando de ese modo una transición hacia el estado en el cual el contador permanecerá mientras se ejecuta la operación siguiente. El término de esta operación siguiente genera de nuevo una señal que aumenta el conteo. Cuando se completa la última operación en el sistema, el estado debe regresar al estado inicial, o ponerse en cero. A lo largo del proceso, mientras se efectúa una operación específica y la máquina está en algún estado particular, la ocurrencia de alguna circunstancia antes de que la operación se complete quizás requiera regresar a un conteo *anterior* en vez de avanzar. El número de operaciones que se van a efectuar determinará el número de módulo del contador.

Contador ascendente-descendente de módulo 6

Consideraremos ahora un ejemplo de contador multimodo. Además del reloj, un contador síncrono tendrá una línea de entrada, x . El conteo se incrementará en uno cuando $x = 0$ y se reducirá en uno cuando $x = 1$. Suponga que se deben controlar seis operaciones, por lo que el número de módulo es 6. Considere, además, que se usará el código progresivo. En este ejemplo, si bien hay una entrada además del reloj, no hay otras salidas además de las del flip-flop.

Observe del código progresivo de la figura 28a que, a partir de cualquier conteo (estado), la cuenta avanza siguiendo $x = 0$ y regresa siguiendo $x = 1$. Dibuje el diagrama de estados (transición) antes de ver la figura 33a; compruébelo después. La tabla de transición se construye con facilidad a partir del diagrama de transición. Constrúyala usted mismo antes de confirmarlo en la figura 33b.

Suponiendo el uso de flip-flops D , las excitaciones son los estados siguientes. El paso que sigue consiste en construir los mapas lógicos para las excitaciones a partir de la tabla de transición; efectúe la anterior antes de revisar la figura 34a. El paso final corresponde a la implementación. Realice ésta y luego verifíquela utilizando la figura 34b.

7 MÁQUINAS DE ESTADO ALGORÍTMICAS

En este capítulo se usaron ya dos herramientas para describir una máquina secuencial: diagramas de estados y tablas de estados. Ambas son útiles. En esta sección describiremos otra herramienta de gran valor. Repase el ejemplo 1 (un circuito de una entrada y una salida) y la figura 2, donde se introdujeron los diagramas de estados. Empezando en cualquier estado, el arribo de una entrada envía a la máquina a uno de los posibles estados siguientes, dependiendo de la entrada. Estas palabras recuerdan el bloque de condición en un diagrama de flujo que representa un algoritmo. En realidad, es posible crear un diagrama que se asemeja a una gráfica de flujo del programa que contiene la misma información transmitida por el diagrama de estados o por la tabla de estados.

Principios básicos

Revise el enunciado del ejemplo 1, así como su diagrama de estados correspondiente, que se consideró antes en el capítulo. Con la máquina en cualquier estado específico, al arribo de un bit de entrada requiere una decisión relativa a la salida que se emitirá y al estado al cual se dirigirá la máquina. En el diagrama de estados de la figura 2, la condición se muestra como una línea dirigida que sale de cada círculo que representa a un estado. Un grafo de flujo debe incluir analogías de

- Círculos que representan estados;
- Líneas dirigidas que conducen a los estados siguientes con salidas especificadas.

Recuerde que un circuito secuencial se conoce como *máquina de estados* (abreviatura para máquina de estados finitos). Puesto que el enunciado del problema para el diseño de una máquina de estados es similar a un algoritmo, la máquina también se conoce como *máquina de estados algorítmica*, o MEA. Es posible construir una gráfica de flujo, llamada gráfica MEA, que describa la operación de una máquina de estados algorítmica. Un diagrama de flujo que describa cualquier algoritmo incluye una caja de condición donde se decide; ésta es la forma familiar de rombo que se muestra en la figura 35a o la variación de ella en la figura 35b, y se le llama *caja de decisión*. Las líneas que salen de una caja de decisión son las *trayectorias de salida*.

En un diagrama de flujo MEA, el círculo que encierra el estado en el diagrama de estados se sustituye por un rectángulo llamado la *caja de estados*, como se muestra en la figura 32c. En lugar de escribir el mismo nombre en el interior, como en el diagrama de estados, se escriben tanto el nombre del estado como el código binario arriba del rectángulo. (En una máquina de

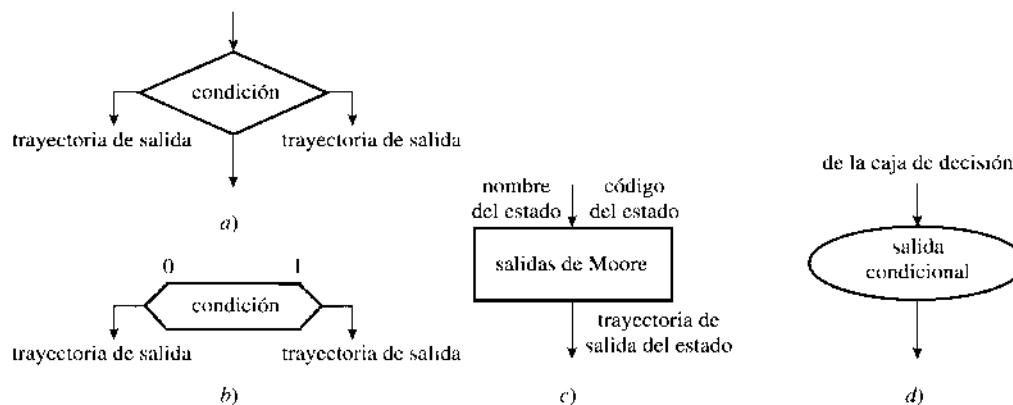


Figura 35. Cajas en diagramas MEA. a), b) Caja de decisión. c) Caja de estado. d) Caja de salida condicional.

Moore, las salidas se asocian con cada estado; por consiguiente, en ese caso, la salida apropiada se escribe dentro de la caja de estados.) Advierta que las dos cajas que se han descrito hasta ahora existen en todos los diagramas MEA.

En máquinas de Mealy la salida depende tanto de la entrada como del estado presente. En este caso se usa otra caja en el diagrama, conocida como *caja de salida condicional*. Para distinguirla de la caja de estados, se emplea una forma oval (figura 32d).

En una máquina de estados, un pulso del reloj inicia la acción, ya sea que esté o no presente una nueva entrada. Empezando en cada estado, deben tomarse decisiones en cuanto a las transiciones de estado y la salida. Una unidad básica en un diagrama MEA puede considerarse compuesta de una caja de estados simple y de todas las demás cajas de decisión y condicionales cuya trayectoria de salida lleva a otro estado. Una unidad de este tipo se denomina *bloque MEA*. De esta manera un diagrama MEA es simplemente una interconexión de estos últimos bloques. Cada estado hacia el cual se efectúa una transición es el punto de inicio de otro bloque. Por claridad, a menudo ayuda encerrar los bloques individuales (utilizando líneas punteadas), aunque hacerlo no agrega nada al diagrama. Las entidades importantes son el estado, la condición y las cajas de salida condicionales; el que se encierran o no estas combinaciones de caja para delinear un bloque MEA es secundario.

EJEMPLO 12

Construyamos un diagrama MEA para el control automático de una puerta de cochera. Sea x la señal de entrada que resulta de un sensor activado mediante un interruptor físico; la salida es z . El valor de x cambia de valor cuando se activa el interruptor. La señal de salida z controla el mecanismo que abre y cierra la puerta de la cochera; z también cambia. Un reloj de baja frecuencia sincroniza el sistema. Suponiendo que está cerrada la puerta de la cochera, cuando se activa el interruptor, la entrada se convierte en 1 y la máquina cambia de estado en el siguiente flanco del reloj. La siguiente vez que se activa el interruptor, la señal se vuelve 0. En respuesta a esta entrada la salida va a 0 en el siguiente flanco del reloj. Un diagrama MEA describe este sistema simple.

La máquina tiene dos estados: abierto y cerrado. Un bloque MEA empezando desde el estado cerrado se muestra en la figura 36a. Éste no indica cómo se alcanzó ese estado. Después de que se recibe la señal de entrada "abierto", se emite la salida 1. Esto quiere decir que la puerta de la cochera debe abrirse, por lo que el siguiente estado es el "abierto". La siguiente vez que se

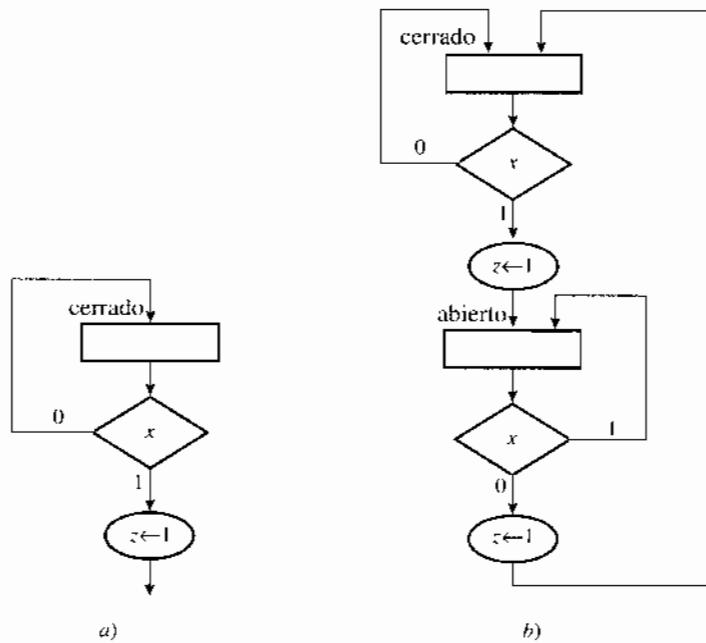


Figura 36. Diagrama MEA para un controlador de puerta de cochera.

actúa sobre el interruptor, la entrada cambia de valor ($x = 0$), y eso mismo sucede con la salida ($z = 0$). El estado “cerrado” se alcanza cuando $x = 0$ mientras se encuentra en el estado abierto. La figura 36b muestra el diagrama terminado. ■

La utilidad de los diagramas MEA no es evidente a partir de este ejemplo sencillo. Para obtener algo más sustancial, volvamos al ejemplo 1 muy al principio de este capítulo, en el cual se describe el detector de secuencia.

EJEMPLO 13

En el ejemplo 1, la salida z se convertirá en 1 sólo después de la recepción de la secuencia de entrada ...0110, independientemente de la secuencia que la preceda. (Vuelva a leer el ejemplo antes de proseguir.) Existen dos posibilidades: ya sea que el bit 0 más reciente en la secuencia anterior forme el primer bit de una secuencia aceptable (la posibilidad de *traslape*), o que la máquina regrese al estado de restablecimiento y empecemos de nuevo. Evidentemente el diseño resultará diferente en los dos casos. El ejemplo 1 tiene que ver con el caso de *traslape*.

Consideremos que el estado S_1 es al que se llega mediante una entrada $x = 0$. Los primeros dos bloques se muestran en el diagrama MEA parcial en la figura 37a. Si la entrada es 0 mientras la máquina se encuentra en cualquiera de los primeros dos estados, la transición será al estado inicial, como se indica. El diagrama MEA terminado se muestra en la figura 37b. En tanto que en el estado S_3 , la recepción de una entrada $x = 1$ estropeará la secuencia aceptable y enviará a la máquina a un estado en el que permanecerá para cada $x = 1$ consecutivo. Escapa de este estado hacia el inicial con una entrada $x = 0$. Sin embargo, si $x = 0$ mientras la máquina se encuentra en el estado S_3 , se emitirá una salida de 1 y regresará también a S_1 . ■

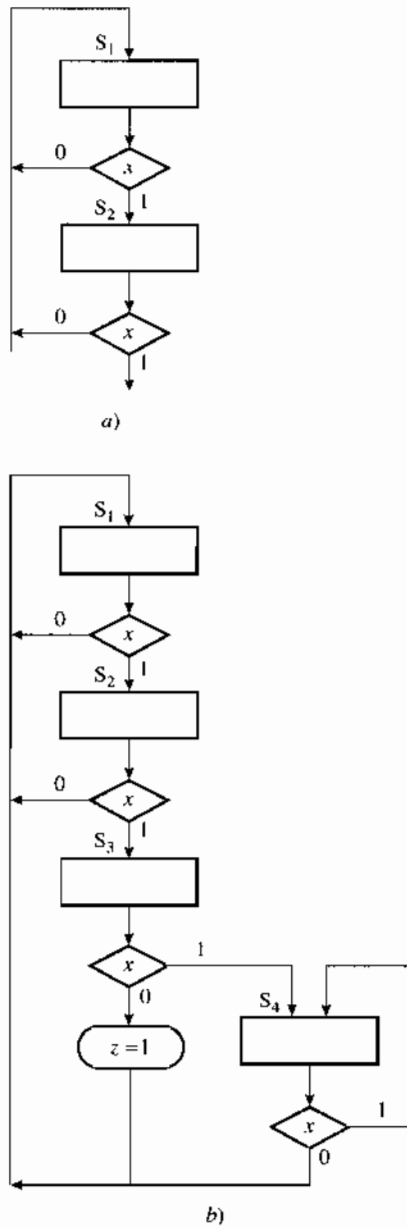


Figura 37. Diagrama MEA para el ejemplo 13.

EJEMPLO 14

Los diagramas MEA son en particular útiles cuando hay un gran número de entradas (el uso de los diagramas de estado resulta irrazonable con tres o más entradas). Considere el diseño de un controlador de semáforo en una intersección que por lo general está muy concurrida durante el día y poco en la noche. Hay una dirección preferente para mantener la luz verde, aunque durante el día la intersección está lo suficientemente concurrida para que no se usen los sensores de detección de automóviles. Durante el día el semáforo pasa por una secuencia fija de verde, amarillo y rojo de la misma duración en ambas direcciones. Durante la noche la señal correspondiente a la dirección preferente permanece en verde a menos que se detecte un automóvil en la dirección no preferente. La dirección preferente es norte/sur.

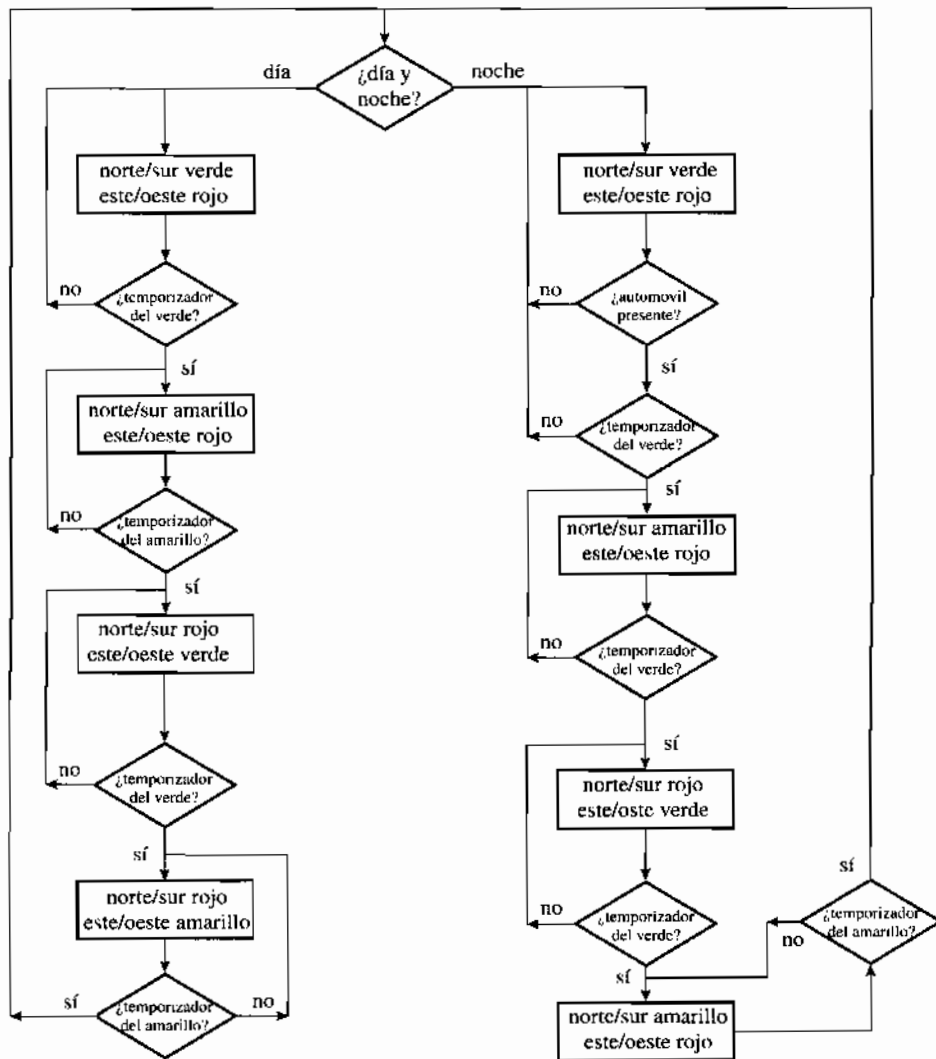


Figura 38. Diagrama MEA para el ejemplo 14.

El controlador de semáforo tiene cuatro entradas: una entrada de sensor para detectar la presencia de un automóvil sobre la vía preferente, una entrada de estado del día para indicar si es de día o de noche, y dos entradas temporizadoras para determinar la duración mínima de la señal verde en una dirección determinada y la duración de la señal amarilla. El controlador cuenta con seis salidas, una para cada señal de color (verde, amarillo y rojo) en cada dirección (norte/sur y este/oeste). El controlador puede considerarse como la ejecución de dos algoritmos dependiendo del estado del día, de manera que la entrada del estado del día determina cuál de los algoritmos se ejecuta. La única diferencia entre la operación durante el día y la noche es que el sensor de presencia de automóviles se usa durante la noche para decidir si se mantiene o no la luz verde en la dirección preferente. El diagrama MEA que describe al sistema se muestra en la figura 38. Antes de que lo revise, trate de crear uno por su propia cuenta y confirme su versión comparándolos. ■

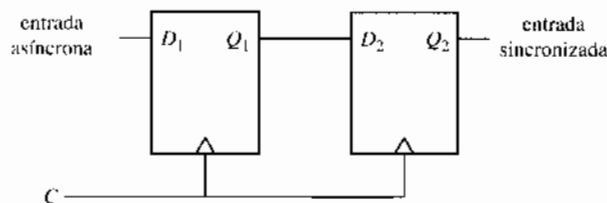


Figura 39. Sincronización de entradas asíncronas para una máquina de estados.

8 ENTRADAS ASÍNCRONAS

Recuerde del capítulo 5 que las implementaciones de máquinas de estados debe satisfacer el tiempo de establecimiento (t_{es}) y el de retención (t_r) de los flip-flops. Las señales de excitación deben ser estables por un periodo de tiempo (t_{es}) antes de la transición de reloj y debe sostenerse por un periodo de tiempo (t_r) después de esta misma. Las señales generadas dentro de la máquina de estados no preocupan, puesto que el retardo de un flip-flop por lo común es más largo que el tiempo de retención, y el tiempo del ciclo del reloj puede incrementarse para asegurar que el tiempo de establecimiento se respete. Si se requiere un tiempo de ciclo más rápido, entonces el nuevo estado y los decodificadores de salida deben rediseñarse para reducir el retardo.

Las señales que se producen fuera de la máquina de estados pueden o no sincronizarse mediante el mismo reloj. Puede tratarse de las señales de un sensor, interruptor, tablero o canal de comunicaciones. Estas señales conmutan en tiempos arbitrarios, por lo que no hay garantía de que respetarán los requerimientos del tiempo de establecimiento y retención de los flip-flops.

Cuando se violan estos requerimientos del flip-flop, la transición de estados resulta impredecible. Es posible que el flip-flop entre a un estado indefinido (un nivel de voltaje entre bajo y alto) durante un breve periodo de tiempo antes de conmutar (impredeciblemente) a un estado definido (0 o 1). El estado indefinido recibe el nombre de *estado metaestable*. Un flip-flop característico permanece en el estado metaestable durante un corto periodo de tiempo, pero es imposible predecir el estado estable que alcanza después de la metaestabilidad. Esto puede provocar que una máquina de estados efectúe una transición de estado errónea.

Para reducir la probabilidad de que una entrada asíncrona provoque una transición de estados errónea, agregar un flip-flop adicional, como se muestra en la figura 39, puede sincronizar las señales. En este circuito es posible que el primer flip-flop entre a la metaestabilidad, aunque con mucha probabilidad alcanzará un estado estable antes del siguiente acontecimiento de reloj. Así, la probabilidad de que la salida del segundo flip-flop tenga un estado válido es muy alta (mucho mayor que si sólo se usara un flip-flop para sincronizar la señal).

Comunicación asíncrona (protocolo de “apretón de manos”)

En ocasiones resulta necesario que dos circuitos secuenciales síncronos con diferentes relojes se comuniquen datos entre sí. Los dos sistemas podrían ser dos computadoras, una computadora y un dispositivo de entrada o salida, o incluso un CPU y memoria. Esta comunicación es asíncrona, por lo que se requiere un protocolo simple para asegurar que los datos se transmitan de manera apropiada. El protocolo que se usa para la comunicación asíncrona se conoce comúnmente como *protocolo de “apretón de manos”*. Cada una de las dos máquinas con reloj independiente M_1 y M_2 que se comunican entre sí utiliza una secuencia de señales para solicitar o enviar datos a la otra máquina.

De ese modo, M_1 y M_2 tienen señales de control y señales de datos que pasan entre ellas. La máquina que inicia la comunicación (digamos, M_1) envía una señal de solicitud de comunica-

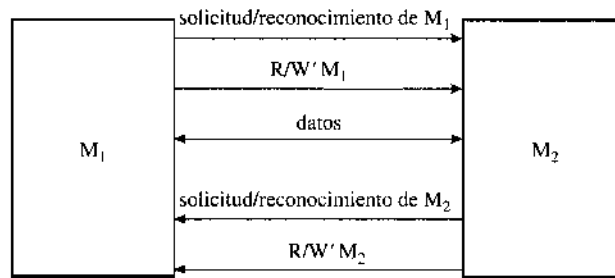


Figura 40. Conexiones requeridas para la comunicación asíncrona entre dos máquinas de estado con relojes independientes.

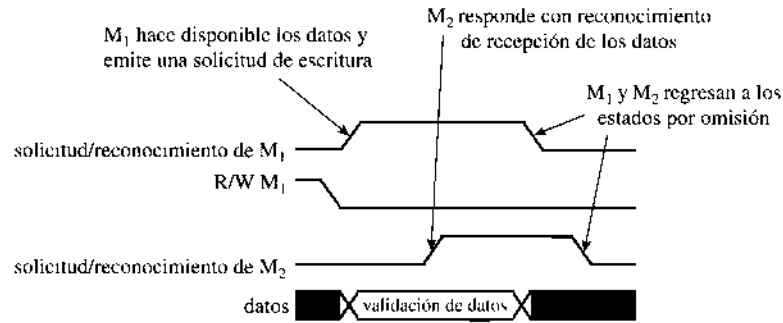


Figura 41. El diagrama de temporización para una operación escrita de M_1 a M_2 .

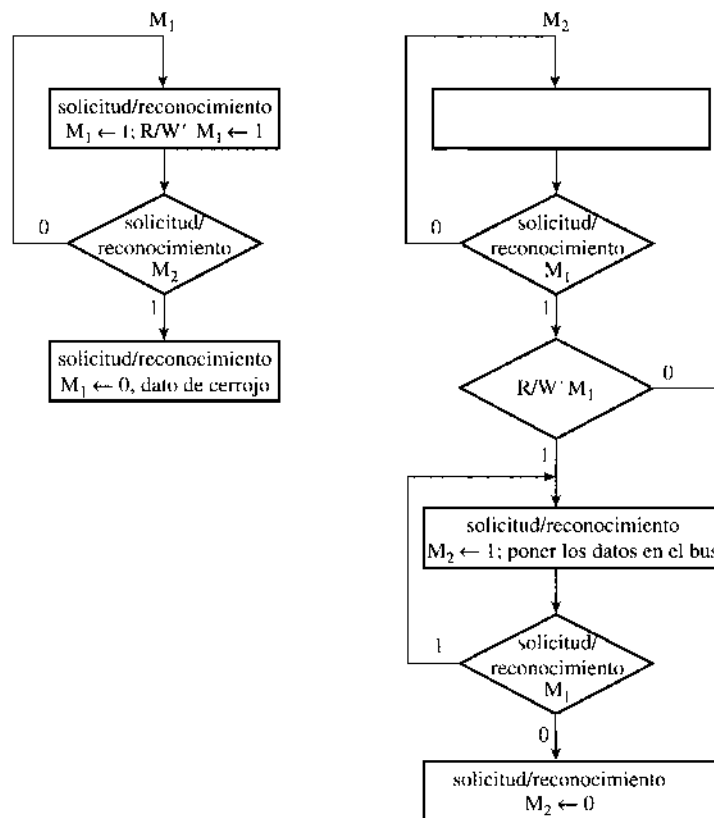


Figura 42. Diagramas MEA para una solicitud de lectura asíncrona de la máquina M_1 a M_2 .

ción a la otra máquina, M_2 ; también envía una señal de control que indica el deseo de leer o escribir datos a la máquina M_2 . En el caso de una solicitud de lectura, M_2 responde con una señal de reconocimiento cuando los datos están listos para M_1 . En el caso de una solicitud de escrituras, M_1 debe hacer que los datos estén disponibles antes de enviar la solicitud a M_2 . M_1 mantiene el dato en la línea de datos hasta que M_2 responde con una señal de reconocimiento. Las comunicaciones de M_2 a M_1 trabajan de manera similar. Las conexiones que se requieren para implementar este protocolo se muestran en la figura 40. El diagrama de temporización correspondiente a M_1 para escribir (o enviar) datos a M_2 se muestra en la figura 41.

Ejercicio 13. Dibuje el diagrama de temporización correspondiente a M_1 para leer (o recibir) datos de M_2 . ♦

La secuencia de señales de control que se requiere para la comunicación entre las máquinas M_1 y M_2 se produce mediante una secuencia de transiciones de estado dentro de las mismas máquinas. Los diagramas MEA para una solicitud de lectura de M_1 a M_2 se muestran en la figura 42.

RESUMEN Y REPASO DEL CAPÍTULO

Este capítulo presentó el diseño de circuitos secuenciales de la variedad síncrona. Para describir cómo se efectúan las transiciones de un estado a otro, creamos las herramientas de los diagramas de estados, tablas de estados y máquinas de estado algorítmicas. Introdujimos la clase de circuitos secuenciales llamados contadores. Los temas incluyeron:

- Modelo de Mealy y modelo de Moore de una máquina secuencial.
- Diagramas de estados, traducción de una especificación de desempeño en un diagrama de estados, con la verificación del diagrama de estados.
- Tabla de estados, construcción de una tabla de estados a partir de un diagrama de estados o directamente de una especificación del problema.
- Diferencia entre un estado de restablecimiento y un estado inicial.
- Asignación de valores binarios a estados.
- Análisis de circuitos secuenciales.
- Transición de estados y tablas de salidas.
- Elección de un tipo de flip-flop en el diseño.
- Mapas de excitación de flip-flop.
- Implementación de circuitos secuenciales a partir de mapas de excitación.
- Distinguibilidad y equivalencia de estados.
- Minimización de máquinas.
- Máquinas con rangos de memoria finita:
 - Máquinas de memoria de entrada finita.
 - Máquinas de memoria de salida finita.
 - Máquinas de memoria finita.
- Contadores síncronos de modo simple:
 - Contadores de distancia unitaria.
 - Contadores de anillo.
- Estados indeterminados.
- Contadores multimodo.
- Máquinas de estado algorítmicas.
- Diagramas MEA.
- Caja de condición.

- Caja de estado.
- Salida condicional.
- Bloque MEA.
- Sincronización de máquinas con entradas asíncronas.
- Comunicaciones asíncronas. Protocolo de "apretón de manos".

PROBLEMAS

Muchos de los problemas que siguen requieren del diseño de una máquina secuencial. El proceso de diseño incluye varios pasos. Cuando estudie el capítulo, quizás desee abordar las secciones anteriores de cada problema antes de estudiar todo lo necesario para completar el diseño. Es posible entonces que regrese a cada problema a medida que aprenda cada paso sucesivo. Guarde las partes previas de la solución conforme avance.

1. Diseñe un circuito secuencial síncrono que tenga una sola entrada x y una sola salida z . La salida z se convertirá en 1 luego de la terminación de la secuencia de entrada 0101, ya sea que ésta forme parte o no de una secuencia traslapada (como 00010101).
 - a. Construya un diagrama de estados y la tabla de estados.
 - b. Construya un mapa de asignación de estados apropiado.
 - c. Suponga el uso de flip-flops JK y construya una tabla de transición.
 - d. Construya mapas de excitación y salida.
 - e. Dibuje el diagrama del circuito resultante.
 - f. Repita las partes c, d y e pero con flip-flops D . Compare la complejidad de los circuitos.
2. La salida z de un circuito secuencial síncrono de una entrada, una salida, se volverá 1 cada vez que la secuencia de entrada sea 1101 o 1001. (Las secuencias traslapadas producirán salidas múltiples.) Efectúe las seis partes del diseño especificadas en el problema 1.
3. Use las mismas condiciones que en el problema 2 salvo que el circuito vuelva a un estado de restablecimiento luego de emitir una salida 1. (¿Qué es lo que ocurre con las secuencias de entrada traslapadas?)
4. Efectúe todos los pasos de diseño señalados en el problema 1 para cada una de las siguientes especificaciones.
 - a. La salida de una máquina de una entrada, una salida, será $z = 1$ si la entrada presente x es la XOR de sus dos valores precedentes.
 - b. La salida es $z = 0$ cuando bits de entrada consecutivos con valor 0 son de longitud par y bits de entrada consecutivos con valor 1, son de longitud impar. La salida será $z = 1$ siempre que exista una discrepancia en este patrón.
 - c. La salida se vuelve $z = 1$ siempre que el bit de entrada sea el producto lógico de sus dos valores previos.
5. La secuencia de entrada de una máquina secuencial de una entrada, una salida, está conformada por palabras de 4 bits consecutivas. Cada palabra es una entidad; las palabras no se forman traslapando secuencias. La salida corresponderá a 1 siempre que el número de bits 1 en una palabra sea impar. Efectúe todas las partes del diseño que se especificaron en el problema 1.
6. Repita el problema 5 con la excepción de que, además de tener un número impar de bits 1, la salida se vuelve 1 sólo si la palabra de 4 bits empieza con un 1. Efectúe todas las partes de diseño que se especificaron en el problema 1.
7. Construya una tabla de estados para el generador de bit de paridad que se describió en el ejemplo 2 directamente de la descripción del problema, sin referencia al diagrama de estados. Compare con la figura 4.
8. Un generador de bit de paridad recibirá mensajes codificados de 4 bits seguidos por un espacio en blanco (un 0). Un bit de paridad de 1 se generará y se insertará en el espacio en blanco si y sólo si la paridad (el número de 1s) de los 4 bits precedentes es impar.

- a. Construya un diagrama de estados y una tabla de estados para este generador de bit de paridad.
 - b. Pruebe su diagrama para verificar que, empezando a partir del estado de restablecimiento, se produce la salida correcta para diversos mensajes de 4 bits.
 - c. Efectúe las partes restantes del diseño especificado en el problema 1.
9. Se ha encontrado que una máquina secuencial tiene tres estados, A, B y C; existen, entonces, sólo dos variables de estado. Especifique tres asignaciones diferentes de palabras de código de 2 bits a los tres estados de manera tal que cualquier otra asignación equivalga a intercambiar las dos variables de estado, invertir cualquier variable, o ambas situaciones.
10. Luego de asignar la combinación 000 al estado A en el ejemplo 4, la adyacencia AG puede satisfacerse al asignar G a una de las otras dos celdas en el mapa de asignación al lado de 001. Elija otra celda para satisfacer esta adyacencia; use después las reglas de adyacencia para obtener una asignación diferente de la del ejemplo 4.
 - a. Complete la implementación, utilizando flip-flops *JK* y compare la complejidad de los decodificadores de estado y salida con la que se obtuvo en el ejemplo 4.
 - b. Encuentre una implementación utilizando flip-flops *D*; compare de nuevo la complejidad.
11. En el ejemplo 6 ubique la tabla de transición de la figura 16c y suponga que la implementación se efectuará con flip-flops *JK*. Construya mapas lógicos para las excitaciones *J* y *K*, determine las funciones de excitación y dibuje el circuito de diagrama secuencial que se produce. Compare la complejidad con la de la implementación utilizando flip-flops *D* en la figura 17.
12. En el ejemplo 6 (el detector de cambio de nivel), considere que las adyacencias que demandan las reglas prácticas son {DE, DF, EF} y {DG, EG, DE}. Sólo pueden obtenerse dos de las adyacencias en cada conjunto. En el ejemplo 6, se eligen las adyacencias no alcanzadas seleccionadas fueron DG y EF. Suponga, en vez de eso, que las adyacencias que se van a considerar sin alcanzar son DF y EG, alcanzándose todas las demás.
 - a. Construya el mapa de asignación resultante.
 - b. Suponga también en este caso que la implementación es con flip-flops *D*; construya las tablas de transición y salida.
 - c. Construya los mapas de excitación para los flip-flops *D*.
 - d. Utilizando los resultados anteriores, dibuje el diagrama del circuito que se genera.
 - e. Compare la cantidad de hardware con el circuito del ejemplo 6 (figura 17).
13. Una máquina secuencial de una salida cuenta con una entrada de datos *x* y con dos entradas de control c_2 y c_1 . La salida igualará la entrada pero con un retraso de uno, dos, tres o cuatro pulsos de reloj, según determine el código de control de entrada $c_2c_1 = 00, 01, 10, 11$, respectivamente. Escriba una expresión para la función de salida y diseñe el circuito, explicando cada paso.
14. Un registro de corrimiento hacia la derecha de entrada en serie, salida en paralelo de 4 bits con preestablecimiento (puesta a 1) asíncrono tiene su estado inicial preestablecido en $y_3y_2y_1y_0 = 1101$, donde y_i es el estado del flip-flop a la entrada del registro. No hay entrada externa con excepción del reloj, y la excitación del registro proviene únicamente del decodificador de estado. La secuencia de salida deseada corresponde a 110111001000; ésta se repite luego de estos 12 bits. Diseñe la lógica combinatoria (decodificador de estado) como un circuito mínimo. En la figura P14 se muestra un diagrama apropiado.

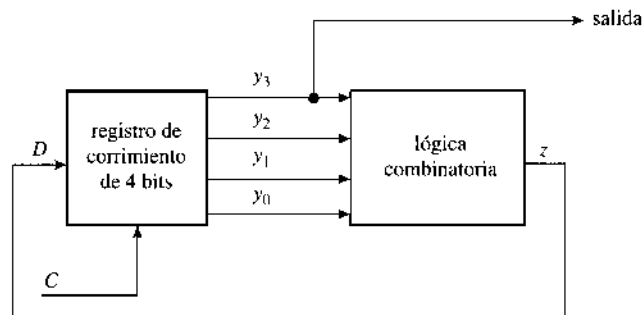


Figura P14

15. Use la misma estructura que la del circuito de la figura P14, pero esta vez con la excitación del registro de corrimiento siguiente:

$$D = y_1y_3 + y_2y_4 + y_3y_4$$

Suponga que el estado inicial del registro de corrimiento se ha fijado en 0101. Determine la secuencia de salida del registro de corrimiento.

16. Diseñe un circuito lógico combinatorio con una sola salida, D . Ésta se convertirá en la entrada a un flip-flop D .
- Suponga que este circuito tiene dos entradas, J y K . Diseñe el circuito de manera que, junto con el flip-flop D , constituya un flip-flop JK .
 - Suponga, en vez de lo anterior, que el circuito lógico combinatorio tiene una sola entrada denominada T . Diseñe el circuito en forma tal que, junto con el flip-flop D , constituya un flip-flop T .
17. Un circuito secuencial síncrono tiene dos líneas de entrada, x_1 y x_2 , y una línea de salida z . La línea de datos es x_1 y x_2 es una línea de restablecimiento. Siempre que $x_2 = 1$, el circuito se pone en cero. Cuando x_2 se vuelve 0, los primeros 4 bits en la línea de mensaje constituyen una palabra de mensaje. La salida se vuelve 1 si el mensaje recibido es 1010. Al final del cuarto bit de cualquier palabra recibida cuando $x_2 = 0$, el circuito entrará en el estado de espera, donde permanece hasta que sea restablecido y donde la salida es 0 para cualesquiera bits de entrada después del cuarto bit.
- Construya un diagrama de estados y una tabla de estados.
 - Lleve a cabo el resto del diseño e implementelo utilizando flip-flops JK .
 - Construya un diagrama de temporización que muestre el reloj y las entradas y salidas.
18. Repita el problema 17 si el mensaje es 1100, utilizando:
- Flip-flops D
 - Flip-flops JK
19. Repita el problema 18 si la longitud de palabra es de 5 bits y el mensaje es 11011.
20. Diseñe una máquina secuencial que tendrá la totalidad de las características siguientes:

Ninguno de 2 bits de salida consecutivos puede ser 1.

El bit de salida no puede ser 1 luego de recibir el segundo de 2 bits de entradas consecutivos 0.

La salida será 1 si cualquiera de 2 bits de entrada consecutivos es 1, a menos que esto entre en conflicto con el primer requerimiento.

Una posible secuencia de entrada/salida, por ejemplo, es la siguiente:

x : 0010101111010000

z : -010101010101000

21. Diseñe un circuito secuencial síncrono de una entrada, una salida, que tenga las siguientes características.

Ninguno de 3 bits de salida consecutivos puede ser 1.

La salida será 1 si, de 3 bits de entradas consecutivos, sólo dos son 1, a menos que esto entre en conflicto con el primer requerimiento.

Una posible secuencia de entrada/salida, por ejemplo, es la siguiente:

x : 000011010111001010110110

z : - - 0001100110100010110110

22. Diseñe una máquina secuencial para la cual la salida se convierte en 1 si y sólo si justo 1 de los siguientes 3 bits es un 1: el bit de entrada presente y los últimos 2 bits de salida.

Una posible secuencia de entrada/salida, por ejemplo, es la siguiente:

x : - - 001011011100101001001

z : 11010100010000110110110

23. Diseñe una máquina secuencial que tiene las siguientes propiedades. Cuenta con dos entradas: x y c , una entrada de control. Cuando $c = 0$, la máquina regresa al estado de restablecimiento A, independientemente de x . Con la máquina en el estado de restablecimiento A, siempre que $c = 1$, los valores de x en los siguientes tres pulsos de reloj constituirán una palabra binaria. Una salida de $z = 1$ ocurrirá en el tercer bit si y sólo si los 3 bits son iguales: 000 o 111. En otro caso la salida será 0. El estado al que entra la máquina en el tercer bit de una palabra de entrada es el de espera. La máquina no abandona este último estado hasta que ocurre $c = 0$, en cuyo caso se restablece.
- Construya un diagrama de estados y una tabla de estados para esta máquina.
 - Si es posible, reduzca esta tabla a una que tenga el menor número de estados.
 - Efectúe los pasos necesarios para llegar a ecuaciones de excitación, suponiendo que la máquina se implemente con flip-flops D. Después dibuje el diagrama de circuito correspondiente.
 - Repita la parte c para el caso en que la máquina se implemente con flip-flop JK.
24. a. Dada la tabla de estados de la figura P24, suponga la siguiente secuencia de entrada con la máquina inicialmente en el estado A:

10011101011001

Determine la secuencia de salida resultante.

- Obtenga una tabla de estados reducida mínima equivalente a la dada, dejando que el nuevo estado A sea el bloque en el cual aparece el estado anterior A.
- Empezando de nuevo en el estado A de la tabla reducida, suponga la misma secuencia de entrada y encuentre la secuencia de salida resultante. Compare el resultado con el de la parte b. ¿Está usted sorprendido?

PS	NS,z	
	$x = 0$	$x = 1$
A	B,0	G,0
B	B,1	H,1
C	F,1	D,0
D	B,0	H,0
E	F,1	D,0
F	F,0	C,1
G	E,0	A,0
H	E,0	A,0

Figura P24

- Diseñe un circuito secuencial síncrono de una entrada, una salida, que produzca una salida de 1 siempre que exista un número impar de 1s en los últimos tres símbolos de entrada.
 - Dibuje varios diagramas de temporización, suponiendo diferentes combinaciones de entradas.
- Diseñe un circuito secuencial síncrono de una entrada, una salida, que genere una salida de 1 siempre que los últimos 4 símbolos de entrada correspondan a un número binario que es
 - Un múltiplo de 3
 - Un múltiplo de 5
 - Un múltiplo ya sea de 3 o de 5
- Los datos que aparecen en una línea sincronizada con un reloj nunca deben tener tres o más 0s consecutivos o cuatro o más 1s consecutivos.
 - Diseñe un circuito secuencial que detectará tales secuencias y generará una salida de 1 siempre que éstas ocurran.
 - Construya diagramas de temporización apropiados para diferentes combinaciones de entradas.
- La salida de un circuito secuencial síncrono será la misma que la entrada pero retrasada por tres de cuatro periodos de reloj bajo el control de una segunda entrada, c . El retardo será tres periodos de reloj cuando $c = 0$ y cuatro periodos cuando $c = 1$.

- a. Diseñe el circuito.
 - b. Construya diagramas de temporización para los dos casos.
29. Un circuito secuencial síncrono tiene dos líneas de entrada, w y x , y una sola línea de salida, z . Considere que $W = w_2w_1w_0$ y $X = x_2x_1x_0$ es una secuencia de 3 bits en las líneas de entrada que representan números binarios, siendo los bits más recientes w_2 y x_2 . La salida será 1 siempre que $w \geq X$.
- a. Diseñe el circuito.
 - b. Dibuje los diagramas de temporización para varias combinaciones de palabras de entrada.
30. Un circuito secuencial síncrono tiene dos entradas de datos a y b , una entrada de control c , y una sola salida z . La salida es 0 salvo que $z = 1$ en cualquiera de dos condiciones:
- $c = 0$ y a y b tuvieron valores idénticos dos periodos de reloj anteriores.
 $c = 1$ y $a = b'$ tres periodos de reloj anteriores.
- a. Diseñe el circuito.
 - b. Construya un diagrama de temporización, mostrando las señales de reloj, entrada y salida para los dos valores de c .
31. Un circuito secuencial síncrono de una entrada, una salida, generará una salida de 1 siempre que x tenga el mismo valor que tuvo durante tres periodos de reloj previos; en otro caso la salida será 0.
- a. Diseñe el circuito.
 - b. Construya diagramas de temporización que muestren las señales de reloj, entrada y salida para los dos casos.
32. Un circuito secuencial síncrono de una entrada, una salida, generará una salida de 1 siempre que ocurra cualquiera de las siguientes secuencias de entrada: 011, 1001, 11011. La salida será 0 en otro caso.
- a. Diseñe el circuito.
 - b. Dibuje diagramas de temporización para las posibles secuencias de entrada, indicando las señales de reloj, entrada y salida.
33. Una máquina de una entrada, una salida, tendrá las siguientes salidas:

$$\begin{aligned} z(t) &= z(t-1)z(t-2) && \text{cuando } x(t) = 0 \\ z(t) &= z(t-3) && \text{cuando } x(t) = 1 \end{aligned}$$

Si esta máquina tiene un rango de memoria finita, especifique su clase y obtenga una implementación canónica.

34. Modifique el problema 33 introduciendo una entrada de control, c . La salida que se especifica en el problema 33 se obtendrá cuando $c = 1$. Cuando $c = 0$, en cambio, la salida será 1 siempre que los últimos 2 bits de entrada sean idénticos. Obtenga una implementación canónica.
35. Determine si cada una de las siguientes máquinas es de memoria finita. En caso afirmativo, señale su tipo (memoria de entrada finita, memoria de salida finita o ninguna de ellas) y su orden.
- a. *Generador de paridad en serie*: La máquina recibe bits de datos serialmente en su entrada e indica en su salida si el número total de 1s recibidos hasta ese punto es par o impar.
 - b. *Sumador en serie*: La máquina tiene dos entradas y recibe dos bits de números binarios en serie en estas líneas de entrada, primero el bit menos significativo. Cuando los bits de peso 2^i (para alguna i) se están recibiendo, la salida corresponderá al bit suma del mismo peso.
 - c. *Multiplicador en serie por una constante*: La máquina recibe serialmente bits de números binarios en su entrada, primero el bit menos significativo, y los multiplica por una constante fija k (que no necesita ser una potencia de 2). Cuando se está recibiendo un bit de entrada de peso 2^i , la salida debe ser el bit producto del mismo peso.
 - d. *Indicador de divisibilidad entre k* : La máquina recibe bits de números binarios en serie en su entrada, primero el bit menos significativo, e indica su divisibilidad entre una constante fija k que no es una potencia de 2. La salida es 1 si y sólo si el número binario recibido hasta ese periodo de reloj (incluyendo la entrada presente) es divisible entre k .

- e. Repita la parte *d* suponiendo que *k* es una potencia de 2.
 - f. Repita la parte *d* suponiendo que el número binario que se recibe primero es el bit más significativo.
 - g. Repita la parte *e* suponiendo que el número binario que se recibe primero es el bit más significativo.
36. Las máquinas secuenciales M_1 y M_2 son de memoria de entrada finita de orden m_1 y m_2 , respectivamente. Una nueva máquina M se obtiene disponiendo en cascada M_1 con M_2 . Esto es, la salida de M_1 es la entrada de M_2 . La entrada y la salida de M son, respectivamente, la entrada de M_1 y la salida de M_2 . Determine si M es una máquina de memoria de entrada finita y, si lo es, establezca su orden.
 37. M_1 y M_2 son máquinas de memoria de entrada finita de orden m_1 y m_2 , respectivamente. Se obtiene una nueva máquina de la manera siguiente. Las entradas de M_1 y M_2 se juntan y constituyen la entrada para M . Las salidas de M_1 y M_2 se llevan por separado y juntas forman la salida de M . Determine si M es una máquina de memoria de entrada finita y, si lo es, obtenga su orden.
 38. Las máquinas secuenciales M_1 y M_2 son de memoria de salida finita de orden m_1 y m_2 , respectivamente. Se construye una nueva máquina M como en el problema 37. Establezca si M es una máquina de memoria de salida finita y, en caso afirmativo, determine su orden.
 39. Los contadores de longitud impar no tienen códigos de distancia unitaria, aunque casi lo hacen, con sólo una transición (usualmente la de pivote a la mitad del conteo) en la cual debe cambiar más de 1 bit.
 - a. Diseñe un contador de anillo de autocorrección de módulo m , donde $m = 5$.
 - b. Repita para $m = 7$.
 - c. Repita para $m = 9$.
 40. El código progresivo es un código de 5 bits para dígitos decimales generados de la manera siguiente. El código para el dígito 0 es 00000. El código para cualquier dígito d_i se obtiene a partir del código para el dígito precedente d_{i-1} fijando primero el msb de d_i igual al complemento del lsb de d_{i-1} , y fijando después los 4 bits inferiores de d_i iguales a los 4 bits superiores de d_{i-1} , en el mismo orden. (Véase la sección en el capítulo 1 acerca de códigos.)
 - a. Utilizando flip-flops D , diseñe un contador síncrono de módulo 10 que cuente en código progresivo; dibuje el circuito.
 - b. Modifique el diseño de manera que el circuito tenga una salida $z = 1$ siempre que el conteo sea 4 o 7.
 41. Un contador multimodo tiene una línea de entrada de pulso x que se sincroniza con el reloj y dos líneas de salida f y g que responden al flanco de ascenso del reloj. Los cambios de nivel en la línea de entrada se separan por al menos cuatro periodos de reloj. La operación del contador se efectúa de la manera siguiente:
 - f se convierte en 1 en cada pulso de reloj.
 - g se convierte en 1 dos pulsos de reloj después.
 - f se vuelve 0 al siguiente pulso de reloj después de que g se convierte en 1.
 - g se vuelve 0 al siguiente pulso de reloj después de que f se vuelve 0.
 - a. Construya un diagrama de temporización que muestre la forma de onda de reloj y las formas de onda de x , f y g .
 - b. Diseñe el contador utilizando un código de distancia 1 y dibuje el circuito.
 - c. Diseñe el contador utilizando el código progresivo, cerciorándose de que éste sea con autocorrección.
 42. El diagrama esquemático de un registro de corrimiento universal con capacidad de corrimiento hacia la izquierda y hacia la derecha se muestra en el capítulo 5, figura 24; éste corresponde a un registro de 4 bits.
 - a. Dibuje un diagrama de transición (un diagrama de estados cuyos estados son códigos ya asignados) para un registro universal de 3 bits. Empezando en cualquier estado, ya sea un 0 o un 1 puede correrse hacia la izquierda o hacia la derecha. Por consiguiente, habrá cuatro arcos que salgan de cada nodo, indicados mediante 0L, OR, 1L y 1R.

- b. Señale cómo se generan una secuencia de contador de anillo estándar de longitud 4 y una secuencia de contador de anillo trenzado de longitud 6 a partir de este diagrama de transición.
- c. Diseñe un circuito decodificador para un registro universal de 3 bits de manera que el circuito combinado con un diagrama esquemático similar al de la figura 24 en el capítulo 5 corresponda a un contador de módulo m , empezando con 000. Considere
- $m = 4$ (dos posibilidades)
 - $m = 5$ (cuatro posibilidades)
 - $m = 6$ (seis posibilidades)
 - $m = 7$ (dos posibilidades)
 - $m = 8$ (cuatro posibilidades)
- d. Dibuje un diagrama de transición para un registro universal de 4 bits y señale cómo se generan una secuencia de contador de anillo estándar de longitud 5 y una secuencia de contador de anillo trenzado de longitud 8 a partir de éste.
- e. Diseñe un circuito decodificador para un registro universal de 4 bits de manera que todo el circuito de la figura 24 en el capítulo 5 sea un contador de módulo m , que empiece con 000. Considere $m = 8$; ¿cuántas secuencias posibles de longitud 8 existen?
- f. Repita *e* para $m = 9$.
43. Suponga que el contador puesto en práctica mediante flip-flops *JK* en la figura 30 se realizará mediante flip-flops *T*, obtenidos a partir de flip-flops *JK* ajustando $J = K$.
- a. Encuentre los mapas de excitación (mapas de *T*) de los tres flip-flops.
 - b. A partir de éstos, determine el decodificador de estados.
 - c. Compare los requerimientos de hardware de esta implementación con aquellos utilizando los flip-flops *JK*.
44. Diseñe una máquina de estados con dos entradas *A* y *B* y una sola salida *C*. La salida se convertirá en 1 sólo si el número de 1s de entrada, desde que la máquina fue puesta en cero, es un múltiplo exacto de 4. No importa en cuál línea de entrada ocurre un 1.
- a. Construya un diagrama de estados. (Conforme avance en esta tarea, piense en las siguientes cuestiones. ¿Es 0 un múltiplo de 4? Con la máquina en un estado particular, ¿qué diferencia en el siguiente estado y salida habría para las entradas $AB = 01$ o 10 ? ¿A qué estado iría la máquina si, habiendo ya recibido tres 1s, la entrada siguiente es $AB = 11$? En un evento de estas características, ¿cuál sería la salida?)
 - b. ¿Cuántas asignaciones diferentes son posibles? Elija una asignación apropiada.
 - c. Suponga el uso de flip-flops *D* y construya los mapas de excitación.
 - d. Escriba expresiones para las excitaciones y la salida.
 - e. Dibuje un circuito que implemente estas expresiones.
 - f. Si desea, intente otra asignación y repita las partes c, d y e. Compare las dos realizaciones. ¿Resultó entretenido?
45. En cada tabla de la figura P45, el objetivo general es construir una tabla mínima reducida equivalente a la original.
- a. Divida los estados de manera que todos ellos sean equivalentes en 1 en una división.
 - b. Refine las divisiones de manera que todos los estados en las nuevas divisiones sean equivalentes en 2.
 - c. Continúe refinando las divisiones hasta que se obtenga una división de equivalencia.
 - d. Construya una tabla de estados reducidos con cada división final como un estado.
 - e. Compare el número de flip-flops necesarios en la implementación tanto de las tablas originales como de las reducidas.
 - f. Implemente cada tabla reducida utilizando flip-flops *JK*.
 - g. Repita *f* utilizando flip-flops *D*.
 - h. Utilizando flip-flops *D*, implemente la tabla de la figura P45b antes de la reducción. Compare el número de flip-flops y la complejidad del circuito para las dos implementaciones.

NS,z			NS,z				
PS	$x = 0$	$x = 1$	PS	$x_1x_2 = 00$	$x_1x_2 = 01$	$x_1x_2 = 11$	$x_1x_2 = 10$
A	A,0	C,0	A	B,0	G,1	C,1	D,0
B	D,1	A,0	B	A,1	E,0	D,1	G,1
C	F,0	F,0	C	H,0	G,1	A,1	C,0
D	E,1	B,0	D	H,0	G,1	C,1	D,0
E	G,1	G,0	E	C,1	H,0	D,1	C,1
F	C,0	C,0	F	D,1	H,0	C,1	G,1
G	B,1	H,0	G	H,1	G,1	A,1	F,0
H	H,0	C,0	H	D,1	E,0	A,1	G,1
a)			b)				

Figura P45

46. Una máquina de estados tiene una sola entrada N y una sola salida D . Mensajes de 4 bits llegan a la entrada. El propósito del circuito es detectar cuándo un mensaje de 4 bits no es una palabra BCD. Esto es, $D = 1$ siempre que la palabra de 4 bits no es un número decimal en código BCD. Suponga que el circuito regresa a su estado inicial (restablecimiento) al final de cada palabra de 4 bits.
- Construya un diagrama de estados y una tabla de estados. (Confirme que su diagrama produce las salidas correctas.)
 - Dividiendo, reduzca la tabla hasta un mínimo.
 - Elija dos palabras de 4 bits, una que no sea un número decimal en código BCD y otra que lo sea. Dibuje diagramas de temporización para estos dos casos.
47. Modifique el problema 46 del modo siguiente. Las palabras de 4 bits no son consecutivas; cuando se recibe el último bit de una palabra, la máquina entra al estado de espera. Mientras se encuentra en este estado, la señal de que viene otra palabra de 4 bits es la aparición de 3 bits 1 consecutivos. Después de recibir el tercer bit 1, la máquina entra al estado de restablecimiento, lista para el siguiente mensaje de 4 bits.
- Construya un nuevo diagrama de estados y una nueva tabla de estados.
 - Reduzca la tabla de estados dividiendo los estados.
48. a. Diseñe un contador BCD síncrono. (Éste podría denominarse contador de módulo 10.) La única entrada es el reloj. Dibuje un diagrama de temporización que incluya el reloj y todas las formas de onda de salida de flip-flop.
- b. Modifique el diseño para un contador que constituirá precisamente una década de un contador BCD decimal. Esto es, cada década representará un dígito decimal en la posición 10^k de un número decimal.
49. Cierta señal binaria consiste en una secuencia periódica de pulsos que tienen el mismo ancho que el pulso del reloj, sincronizados con este último. En cierta aplicación, se espera que el número de bits en una secuencia de 0s sea impar y que el número de 1s sea par. Diseñará una máquina de estados para detectar errores a partir de esta configuración. Esto es, $z = 1$ siempre que se detecte una secuencia par de 0s o una secuencia impar de 1s.
- Construya un diagrama de estados y una tabla de estados. (Piense cómo la máquina sabrá que ha finalizado una secuencia de bits similares.)
 - Si su tabla no es mínima, redúzcala.
 - Efectúe una asignación de estados "óptima" y construya una tabla de transición.
 - Suponga el uso de flip-flops D y escriba expresiones para las excitaciones.
 - Dibuje un diagrama de circuito que implemente estas expresiones.
50. Un circuito secuencial síncrono tiene dos líneas de entrada, x_1 y x_2 , y dos líneas de salida, z_1 y z_2 . En cada pulso del reloj, la combinación x_1x_2 constituye un número binario de 2 bits. Si el valor presente

del número binario es menor que su valor inmediatamente anterior, entonces las salidas son $z_1 z_2 = 10$. Si el valor presente es mayor que el valor precedente, entonces $z_1 z_2 = 01$. Si es el mismo, $z_1 z_2 = 00$.

- a. Diseñe el circuito.
 - b. Dibuje diagramas de temporización para los tres casos.
51. Se espera que palabras de cinco bits que llegan a una línea sean mensajes en código 2 de 5. Sin embargo, quizá existan errores. Diseñe una máquina síncrona cuyo salida es 1 sólo cuando se recibe el quinto bit y la palabra completada no es una palabra válida en código 2 de 5. Las palabras de 5 bits son consecutivas; tan pronto como una palabra de 5 bits se completa, el circuito debe estar listo para recibir el primer bit de la palabra siguiente.
- a. Construya un diagrama de estados. (Sugerencia: ¿A cuántos estados distintos puede hacer el circuito una transición para cada bit entrante después del primero?)
 - b. Construya una tabla de estados.
 - c. Realice una asignación de estados apropiada y construya una tabla de transición.
 - d. Suponiendo el uso de flip-flops *D*, obtenga expresiones para las funciones de excitación y salida.
 - e. Construya diagramas de temporización para el reloj, los bits de entrada y la salida resultante.
52. a. Una máquina secuencial síncrona tiene una línea de entrada x y una línea de salida z . Se pretende que la máquina reciba un número binario de longitud desconocida en la línea de entrada, siendo el primero el bit menos significativo, y que indique en z su divisibilidad entre 5. Esto es, en cualquier tiempo t , $z(t) = 1$ si y sólo si el número binario $x(t) \dots x(0)$ es divisible entre 5. Construya una tabla de estados para una máquina de estas características y minimice el número de estos mismos.
- b. Generalice la parte a: si se va a detectar la divisibilidad entre un número p , donde p es una constante conocida, determine un límite superior en términos de p del número de estados necesarios en la máquina.
 - c. Repita la parte a suponiendo que se recibe primero el bit más significativo.
 - d. Repita la parte b suponiendo que se recibe primero el bit más significativo.
53. La figura P57 muestra un diagrama esquemático de un contador síncrono de módulo 10 cuyos estados son 0-9. Las salidas $Q_3 \dots Q_0$ representan el conteo. Si CH ("habilitar conteo") es 1, entonces el contador se incrementa al siguiente estado en el pulso de reloj que sigue. En otro caso retiene el conteo presente. La salida TC es 1 si y sólo si el conteo es 9.

El objetivo de este problema es diseñar un contador de módulo 10^{16} encadenando en conjunto varios contadores de módulo 10. Se afirma que las entradas CH a los contadores de módulo 10 en la cadena son análogas a las entradas de acarreo de los sumadores completos de un sumador multibit y, en consecuencia, es posible utilizar los principios de acarreo anticipado al diseñar el contador de módulo 10^{16} .

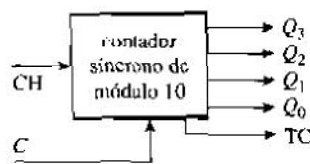


Figura P53

- a. Deduzca las expresiones de generación y propagación para un contador de módulo 10.
 - b. Suponga que se dispone con unidades de anticipación de 4 bits del tipo que se muestra en la figura 8 del capítulo 4, con entradas P, P', G, G' y C_1 y salidas C_2, C_3 . Utilizando éstas y los contadores de módulo 10 (y ninguna otra lógica) diseñe un contador con anticipación de módulo 10^{16} .
 - c. Obtenga un contador de módulo 10^{16} sustituyendo las unidades de anticipación en la respuesta para b por únicamente compuertas AND.
54. a. Diseñe un contador binario de módulo 2 (2 bits) utilizando flip-flops *D*. El contador efectúa la secuencia 00 01 10 11 00... La máquina también tiene una línea de salida que emite un 1 en la cuenta 11.
- b. Dibuje un diagrama de temporización apropiado.

55. Una máquina de estados tendrá una sola línea de entrada y una sola línea de salida. La salida se mantendrá en 0 hasta que ocurra el último bit de las secuencias ...0000 o ...1111, a cuyo tiempo la salida se vuelve 1.
- Construya un diagrama de estados y una tabla de estados; reduzca entonces la tabla hasta una que tenga el menor número de estados.
 - Construya un diagrama MEA.
 - Considerando la implementación con flip-flops *D*, construya las tablas de transición y, a partir de éstas, construya una implementación del circuito.
56. Cierta máquina de estados tendrá la función de determinar cuándo una secuencia entrante de 7 bits no es un código biquinario para un dígito decimal. La máquina tiene dos entradas: DATO y CONTROL. DATO consta de palabras de 7 bits que representan los dígitos decimales en el código biquinario. CONTROL es una señal que inicia la inspección del DATO. Cuando CONTROL = 0 para uno o más pulsos del reloj, la salida permanece en 0. Cuando CONTROL se vuelve 1 y se mantiene en 1, la máquina va a examinar los siguientes 7 bits en DATO. Mientras tanto la salida se mantiene en 0; se convierte en 1 sólo si el séptimo bit completa una palabra que no es un dígito decimal en código biquinario.
- Construya una tabla de estados para esta máquina.
 - Suponiendo el empleo de flip-flops *D*, construya las tablas de transición. Mediante estas tablas, diseñe una implementación del circuito.
57. Diseñe comparador secuencial, con dos líneas de entrada x y y y una sola línea de salida z . $X(x_n, x_{n-1}, x_{n-2})$ es una palabra de 3 bits en la línea x y, de manera similar, $Y(y_n, y_{n-1}, y_{n-2})$ es una palabra de 3 bits en la línea y . Considerando X y Y como números binarios de 3 bits, la salida será 1 sólo si $X \geq Y$.
- Construya un diagrama de estados y una tabla de estados para esta máquina.
 - Suponga que se van a utilizar flip-flops *D*. Construya tablas de transición y, a partir de éstas, una implementación del circuito.
 - Repita *b* utilizando flip-flops *JK*.
 - Alguien sugiere la implementación del circuito con dos registros de corrimiento que leen en paralelo y cierta lógica combinatoria. Lleve a cabo esta sugerencia.
58. El objetivo de este problema es diseñar un contador modelo de Moore ascendente-descendente de módulo 8. (Módulo 8 significa que la máquina cuenta de 0 a 7 en binario. "Ascendente-descendente" significa que cuando el conteo avanza desde 7 (111) va a 0 (000) y cuando desciende desde 0 va a 7.) Además del reloj, la máquina tendrá una salida, x . Cuando $x = 0$, el conteo disminuirá en 1 desde su valor presente y, cuando $x = 1$, se incrementará en 1 desde su valor presente, ocurriendo ambos con el pulso del reloj. Suponga que se van a utilizar flip-flops *D* y que no hay decodificador de salida, siendo los estados de las salidas del flip-flop las que se toman como un número binario.
- Dibuje un diagrama que muestre tres flip-flops y un decodificador de estados como un rectángulo. (¿Es posible identificar la naturaleza de esta máquina?)
 - Construya directamente una tabla de transición en vez de utilizar nombres arbitrarios para los estados y realizar una posterior asignación de estados; utilice los valores binarios del conteo para identificar los estados presente y siguiente.
 - Construya mapas lógicos para cada estado siguiente.
 - Diseñe el decodificador de estados y complete la implementación.
 - Utilizando tiempos arbitrarios de cambios de entrada relativos al reloj, dibuje diagramas de temporización que muestren los pulsos de reloj, la entrada y las salidas de flip-flop.
59. El objetivo es diseñar un contador ascendente-descendente de módulo 8 con una sola entrada x y tres líneas de salida. El número binario representado por la salida $z_2 z_1 z_0$ es el conteo. Este se incrementará en 1 cuando $x = 1$ y disminuirá en 1 cuando $x = 0$. Diseñe el circuito.
60. El propósito es diseñar un contador descendente binario de 3 bits sin otras entradas más que el reloj. En cada pulso del reloj, el contador realiza ciclos a lo largo de la secuencia (000, 001, 011, 101, 100, luego de lo cual repite la secuencia. Los otros dos estados posibles no ocurren.

- a. Utilizando los códigos de estado como "nombres" de estado, construya directamente una tabla de estados. Decida cómo manejar las entradas correspondientes a los renglones de combinaciones que no ocurrirán.
- b. Suponiendo el uso de flip-flops D , el siguiente estado para cada posición en cualquier renglón es el mismo que el valor requerido de D . Construya mapas lógicos para el valor requerido D en términos de los estados presentes. A partir de éstos, escriba una expresión para cada D .
- c. Construya el diagrama de circuito para implementar el contador.
- d. Dibuje un diagrama de temporización que muestre las formas de onda para el reloj y para las salidas de los tres flip-flops.
- e. Suponga ahora que se recurrirá a flip-flops biestables (T). A partir de los requerimientos de excitación para los flip-flops T de la figura 17 del capítulo 5, construya nuevos mapas lógicos para cada T , así como un diagrama de circuito que implemente al contador. Compare esto con la implementación utilizando flip-flops D . Indique si habrá algún cambio en los diagramas de temporización.
61. a. Repita el problema 60 si la secuencia del contador fuera la siguiente: 000, 010, 001, 011, 101, 100, 000, ...
- b. Repita el problema 60 si la secuencia del contador fuera la siguiente: 000, 011, 111, 101, 001.
62. La tabla de estados de la figura P62a se implementará con dos flip-flops Lemon.

PS	NS, z		PS	NS, z	
	$x = 0$	$x = 1$		$x = 0$	$x = 1$
A	C,0	B,1	A	C,0	B,0
B	A,0	A,0	B	A,1	C,1
C	A,0	D,0	C	B,0	D,0
D	C,0	A,1	D	C,1	C,0
	a)		b)		

Figura P62

- a. Utilizando los resultados del problema 18 en el capítulo 5, especifique todas las asignaciones de estado posibles, justificando su respuesta.
- b. Elija una de las posibles asignaciones y efectúe una implementación de circuito.
- c. Repita la parte b para una asignación diferente. ¿Hay razones para elegir una implementación posible en vez de la otra?
- d. Repita cada parte para la tabla de estados que se muestra en la figura P62b.
63. Un contador tendrá una sola entrada de control de 1 bit C . Cuando $C = 0$, el contador de 3 bits efectuará una secuencia a través del código binario. Cuando $C = 1$, efectuará una secuencia a través del código Gray.

Código binario: 000 001 010 011 100 101 110 111 \rightarrow 000

Código gris: 000 001 011 010 110 111 101 100 \rightarrow 000

Las únicas salidas son las de flip-flop que representan los estados.

- a. Construya un diagrama de estados, indicando los estados mediante sus códigos de 3 bits. Indique las transiciones hacia los siguientes estados apropiados para cada C .
- b. Dibuje un diagrama MEA para el contador.
- c. Suponga que el estado presente es 000 cuando la entrada de control sigue la siguiente secuencia.

$C: 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1$

Construya una tabla con la entrada como columna 1, el código de estado presente como columna 2 y el código del siguiente estado como columna 3.

- d. Dibuje un circuito que implemente el contador.

64. Un circuito secuencial síncrono tiene una sola salida z y dos entradas, x y r . La salida es una versión retardada de la entrada x bajo el control de r . Cuando $r = 1$, la salida es igual a lo que era la entrada x tres periodos de reloj antes. Cuando $r = 0$, la salida es igual a lo que era la entrada x dos periodos de reloj antes. Diseñe el circuito utilizando flip-flop apropiados. Indique y explique todos los pasos intermedios.
65. Una máquina de estados tiene una sola salida y —además del reloj— tres entradas: una entrada de datos x y las salidas de un contador de módulo cuatro c_1 y c_0 . La salida de la máquina igualará la entrada de datos pero retardada por un número de periodos de reloj determinado por el conteo c_1c_0 : el retardo en la salida es un periodo de reloj en el conteo 00, dos periodos de reloj en el conteo 01, etcétera.
- Recurra a lo que necesite (diagrama de estados, tabla de estados, mapas lógicos, etc.) para llegar a una expresión correspondiente a la salida. Explique su razonamiento.
 - Encuentre una implementación del circuito.
66. Un cliente ha pedido a su taller de diseño de ingeniería una máquina secuencial síncrona de una entrada, una salida. La salida se volverá 1 siempre que, empezando en algún tiempo, el número de 1s de entrada exceda al número de 0s de entrada. Un ejemplo es:
- x : 0 1 1 0 1 1 0 0 ...
 z : 0 0 1 0 1 1 1 0 ...
- Dé un argumento de por qué es imposible una máquina de estas características o b) construya un diagrama de estados de una máquina que satisfice este requerimiento. En el último caso, construya la tabla de estados e impléméntela.
67. Un circuito secuencial síncrono tendrá tres entradas: A , B y C . La única salida z será 0 excepto para las siguientes entradas posibles:
- $z = 1$ cuando $C = 0$ y A y B tuvieron valores idénticos dos periodos de reloj antes.
 $z = 1$ cuando $C = 1$ y A y B tuvieron valores opuestos tres periodos de reloj antes.
- Obtenga una implementación del circuito y explique su naturaleza.
68.
 - En el ejemplo 7, utilice las expresiones para las excitaciones de flip-flop y la salida z en la página 218 para construir una realización del circuito.
 - Analice el circuito resultante para verificar las expresiones originales.
 - Si estas expresiones no se pueden verificar mediante su circuito, repita la parte *a* hasta que se logre la verificación.
69. Diseñe un contador ascendente-descendente de módulo 8. El conteo aparecerá en un código BCD en tres líneas de salida como $z_2z_1z_0$.

Capítulo 7

Máquinas secuenciales asíncronas

El tipo de circuito secuencial tratado hasta ahora utiliza un reloj para sincronizar todas las transiciones de flip-flop. La velocidad de operación en un circuito síncrono depende de la frecuencia del reloj. Ésta debe ajustarse para aceptar los subsistemas más lentos en el sistema digital completo cuya temporización se debe controlar. Los subsistemas que podrían estar operando a una velocidad superior —respondiendo a cambios en la entrada a medida que éstos ocurren, en vez de esperar el pulso de reloj— resultan por ello frenados de manera innecesaria. Para aumentar la velocidad total, en ocasiones es deseable permitir que algunos subsistemas operen a su propia velocidad —asíncronamente, no en sincronía con el reloj.

Por otro lado, suponga que hay dos sistemas digitales, cada uno sincronizado con su propio reloj, entre los cuales se necesita cierta comunicación. ¿A qué frecuencia de reloj debe efectuarse una comunicación de este tipo si se realiza sincronizadamente?¹ También con estos fines, un sistema asíncrono podría demostrar su utilidad.

En este capítulo consideraremos circuitos secuenciales que no usan una fuente de temporización para sincronizar transiciones de estado. Éstos entran en la clase genérica de los circuitos secuenciales *asíncronos*.² En realidad, ya se han considerado estos circuitos secuenciales. Los cerrojos y las interconexiones de éstos entran también en la clase de circuitos secuenciales asíncronos y conforman los circuitos más comunes de este tipo. Surge una pregunta: puesto que el diseño de esta clase especializada de circuitos ocupará la atención de sólo unas cuantas personas, ¿por qué dedicar tiempo a estudiarlo? Una razón es que muchos conceptos importantes en los circuitos digitales, algunos de los cuales resultan también válidos para los circuitos combinatorios, aparecen en este tipo de estudio.

Además, nunca se sabe en qué punto de la vida profesional algunas ideas fundamentales específicas se volverán útiles. Desistir de este estudio equivale a estrechar nuestra visión. Se encuentran en discusión ideas fundamentales, no los últimos “cómos”, aspectos que de cualquier forma cambiarían rápidamente, puesto que lo contemporáneo hoy no lo será mañana. Por otro lado, debido a que este tema puede considerarse especializado y no de importancia central para lo que resta del libro, es posible tratar todo el capítulo como opcional. Si se omite, sin embargo, harán falta muchas ideas importantes, como ya se señaló.

¹ En muchos grandes sistemas, por lo general, se vuelve problemático el uso de un solo reloj para sincronizar todas las transiciones. En tales casos, es común dividir el sistema completo en subsistemas, cada uno sincronizado por medio de un reloj local.

² Éstos en ocasiones se denominan circuitos secuenciales *retroalimentados*.

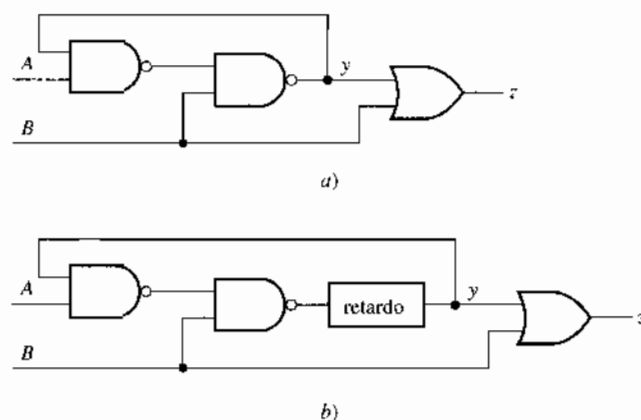


Figura 1. Un circuito de compuertas con retroalimentación.

1 EL MODELO DEL MODO FUNDAMENTAL

En un tipo de circuito asíncrono, las señales de entrada y salida son *niveles*, y las entradas de nivel pueden cambiar en cualquier tiempo. En lugar de utilizar flip-flops para establecer los estados, el inevitable retardo de señales que se propagan a través de las compuertas se utiliza por medio de retroalimentación. Para ver cómo es posible la acción secuencial sin flip-flop, considere el circuito que se muestra en la figura 1a, el cual incluye un lazo de retroalimentación. Cada una de las compuertas en el lazo introduce cierto retardo. Un posible modelo para representar los retardos en las compuertas tiene la finalidad de mostrar el retardo total de todas las compuertas agrupadas en un lugar en el lazo, como en la figura 1b, consideradas las compuertas ideales. La estructura completa puede describirse entonces como un circuito combinatorio con un lazo de retroalimentación que contiene retardo.

La explicación precedente conduce al modelo generalizado que se indica en la figura 2. La totalidad de todas las entradas a la lógica combinatoria recibe el nombre de *estado total*. Lo conforman dos componentes:

$\{x_i\}$, las entradas *primarias* o el *estado de la entrada*.

$\{y_i\}$, las entradas *secundarias*, o el *estado interno*.

Las últimas son las variables de retroalimentación que siguen al retardo. La combinación de las entradas primaria y secundaria genera:

$\{z_i\}$, las *salidas* al mundo exterior.

$\{Y_i\}$, las *excitaciones* que, después de un retardo, se convierten en las entradas secundarias.

Las entradas primarias pueden cambiar en cualquier tiempo de un nivel a otro, pero suponemos que *dos entradas no cambian simultáneamente*. Cuando las entradas han permanecido estables por algún tiempo (mayor que el retardo de propagación a lo largo de cualquier trayectoria en el circuito), no ocurre ningún cambio de señal, y afirmamos que el circuito se encuentra en el *estado estable*. En este caso $y_i = Y_i$ para toda i . Esto es, el estado interno y las excitaciones son iguales.³

En respuesta a un cambio de entrada de nivel, la lógica combinatoria genera un nuevo conjunto de excitaciones, y el circuito entra en un *estado inestable* en el cual $Y_i \neq y_i$. Después de un

³ Son posibles tanto el modelo asíncrono de Mealy como el de Moore. La diferencia de acuerdo con la figura 1 en el capítulo 6 es que están ausentes el decodificador de estado y el reloj. No se establecerán distinciones aquí.

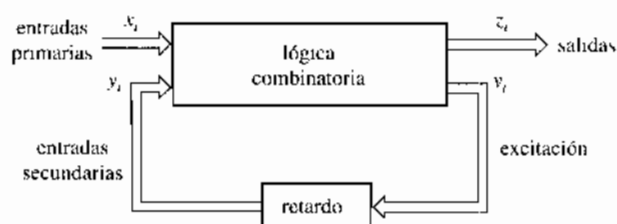


Figura 2. Modelo general de circuitos secuenciales en modo fundamental.

retardo, las secundarias toman sus nuevos valores. Si ocurriera otro cambio de entrada mientras el circuito está en un estado inestable, el estado estable final se volvería incierto. Puesto que una incertidumbre de este tipo resulta intolerable, suponemos que cualquier cambio en los niveles de entrada están separados por más tiempo que el retardo de propagación a través de la trayectoria más larga en el circuito. Los circuitos secuenciales que funcionan de la manera descrita se dice que operarán en el *modo fundamental*.

2 LA TABLA DE FLUJO

El diseño de circuitos asíncronos en modo fundamental es análogo al de los circuitos secuenciales síncronos, pero sin preocuparse por los flip-flops. En el caso síncrono, un paso previo en el proceso es la traducción de una especificación de un problema en una tabla de estados. Un paso similar se efectúa para los circuitos asíncronos en modo fundamental, salvo por la complicación agregada de los estados inestables. A pesar de eso, como consecuencia de los cambios del nivel de entrada, las transiciones de un estado estable a otro, por medio de un estado inestable, se indican también en una tabla.

Tablas de flujo primitivas

La manera más simple para describir el procedimiento relativo a la construcción de una tabla se ubica en el contexto de un problema de diseño. Suponga que un circuito secuencial asíncrono tendrá dos entradas y una salida. La salida será 0 hasta que la entrada se vuelva $x_1x_2 = 11$, pero sólo si sigue a una entrada de 10. (¿Por qué sería imposible que la entrada se convirtiera en $x_1x_2 = 11$ después de un 00?) Como ejemplo, las especificaciones se satisfacen con la siguiente secuencia de salida como resultado de la secuencia de entrada dada:

x_1x_2 :	00	01	11	10	11	10	00	10	11	01	11
z :	0	0	0	0	1	0	0	0	1	0	0

Observe que z no se convierte en 1 debido precisamente a que la entrada se ha vuelto 11; sólo se alcanza el 11 a partir del 10. Un diagrama de temporización que ilustra estas secuencias de salida de entradas se proporciona en la figura 3.

Necesitamos un mecanismo para seguir la traza del "flujo" de los cambios de entrada que producen cambios de excitación, lo cual conduce luego a los cambios de estado. Esto se efectúa mediante una tabla, conocida como *tabla de flujo*, de acuerdo con la figura 4 para este ejemplo.

Al inicio, con ambas entradas 0, el circuito se encuentra en un estado estable denominado (a). El círculo alrededor del nombre del estado indica que éste es estable. Un estado inestable se señala nombrando simplemente el estado sin el círculo. Suponga ahora que x_1 se convierte en 1, de manera que la entrada se vuelve $x_1x_2 = 10$. (Éste es un cambio en una sola entrada.) A la larga, el circuito entrará al estado estable, llamado (b) en la figura 4b; sin embargo, este estado es

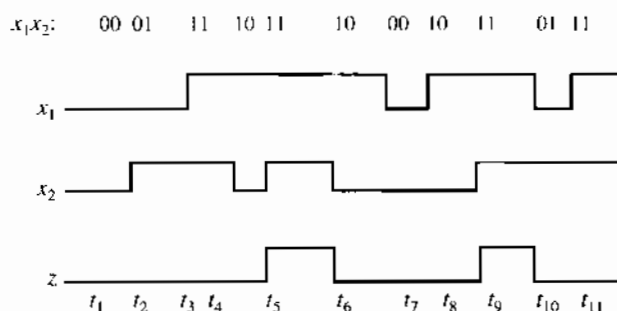


Figura 3. Secuencias posibles de entradas y salida.

Estado, salida x_1x_2				S, z x_1x_2			
00	01	11	10	00	01	11	10
a) $\textcircled{a}, 0 \leftarrow$ salida inicial \uparrow estado estable inicial				b) $\textcircled{a}, 0$ $\textcircled{b}, 0$			
S, z x_1x_2				S, z x_1x_2			
00	01	11	10	00	01	11	10
$\textcircled{a}, 0$			$\textcircled{b}, 0$	$\textcircled{a}, 0$	d	—	b
		c		a	—	c	$\textcircled{b}, 0$
	d	$\textcircled{c}, 1$		—	d	$\textcircled{c}, 1$	b
	$\textcircled{d}, 0$	e		a	$\textcircled{d}, 0$	e	—
		$\textcircled{e}, 0$		—	d	$\textcircled{e}, 0$	b
c)				d)			

Figura 4. Construcción de una tabla de flujo primitiva.

inestable antes de que eso ocurra. Esta secuencia se muestra en la figura 4b; ésta pasa de un estado estable en una entrada a un estado inestable cuando cambia la entrada. En ese caso la transición es hacia el estado estable correspondiente (después de cierto retardo pero sin ningún cambio adicional en la entrada). Esta vez la secuencia es de una columna a otra en el mismo renglón, seguida por un movimiento de un renglón a otro en la misma columna. Lo último ocurre después de cierto retardo. También se indica la salida correspondiente a la entrada específica.

Suponga que x_2 se convierte después en 1, por lo que $x_1x_2 = 11$. Esta vez, a partir del estado estable \textcircled{b} nos movemos primero horizontalmente hacia la columna correspondiente a $x_1x_2 = 11$; el estado inestable se marca con la letra c. Después de un retardo, nos movemos verticalmente hacia el estado estable \textcircled{c} que se muestra en la figura 4c, en cuyo punto se emite una salida de 1, de acuerdo con el enunciado del problema. Luego de esto, un cambio de entrada de $x_1x_2 = 11$ a 01 conduce, después de la estabilización, a otro estado, \textcircled{d} . Situado en el estado \textcircled{d} (columna $x_1x_2 = 01$), suponga que x_1 cambia a 1 de manera que x_1x_2 se convierte en 11. Según las especificaciones del problema, esto no debe dar lugar a $z = 1$. Por consiguiente, el estado

				S, z			
				$x_1 x_2$			
				00	01	11	10
S, z				(a), 0	b,	—	e,
$x_1 x_2$				a,	(b), 0	c,	—
00	01	11	10	—	f,	(c), 0	d,
(a), 0	b,	—		a,		g,	(d), 1
	(b), 0	c	—	a,	—	g,	(e), 0
—		(c), 0	d,	a,	f,	g,	—
			(d), 1	—	(f), 0	(g), 0	e,

a)

b)

Figura 5. Construcción de la tabla de flujo primitiva para el ejemplo 1.

estable que se alcanza por medio de este cambio de entrada no debe ser (c), donde la salida es 1; debe ser algún otro estado, digamos (e). Estas variaciones son también como se indica en la figura 4c.

La tabla de flujo puede completarse de esta manera. Sin embargo, puesto que no son posibles los cambios simultáneos en las dos entradas, no es factible que la entrada cambie de 00 a 11 o viceversa. (¿Qué ocurre de 01 a 10 o viceversa?) Para completar la tabla, a partir de cada estado estable se supone cualquier cambio de entrada *permisible* y se registra la transición apropiada, ya sea como un estado ya anotado en la tabla o como un estado nuevo. Para cada uno de estos últimos se crea un nuevo renglón y, en la columna correspondiente al símbolo de entrada, se anotan el nuevo estado estable y la salida correspondiente. Cuando no se permite que ocurra un cambio de entrada particular, la transición de estado y la salida no están especificadas, lo que se indica por medio de una raya en la columna y el renglón apropiados. La tabla de flujo completa se muestra en la figura 4d. Esto lleva a la siguiente definición:

Una tabla de flujo primitiva es aquella que tiene únicamente un estado estable por renglón, con salidas especificadas sólo para estados estables.

Note las características de la tabla de flujo primitiva en la figura 4d y compárelas con la tabla de estado correspondiente a una máquina síncrona. En primer lugar, la salida para los estados inestables no se ha especificado. Esta omisión se abordará en la sección siguiente.⁴ Una segunda diferencia parece ser que, a pesar de que hay una columna para cada combinación de entrada, no hay columna que liste el "estado presente" que identifique cada renglón. Sin embargo, cada renglón se identifica con un estado estable particular que puede considerarse el "estado presente" correspondiente a este renglón. No hay nada que evite el uso de los nombres de los estados estables como los encabezados de los renglones de la tabla.

Ejercicio 1. Reconstruya la tabla de flujo de la figura 4, proporcionando una columna a la izquierda cuyas entradas de renglón son los nombres de los estados estables correspondientes a cada renglón. Para simplificar, no encerraremos en círculos los nombres de los "estados presentes" en esta columna.

⁴ Puesto que sólo se especifica una salida en cada renglón de una tabla de flujo primitiva, resulta concebible que la salida pueda ser derivada de cada estado y ubicada en una columna de salida independiente. Sin embargo, no lo haremos de ese modo debido a que habrá ocasiones en las que desearíamos asignar salidas diferentes a estados inestables diferentes en un renglón determinado, cuando procedamos a completar la tabla.

Otra característica del todo diferente en comparación con las tablas de estado para máquinas síncronas es que algunas entradas (tanto estados como salidas) no están especificadas. Esto, sin embargo, no constituye una diferencia fundamental. También existen máquinas síncronas especificadas de manera incompleta. Aunque no las estudiamos cuando analizamos este tipo de máquinas en el capítulo 6, subsanaremos esa deficiencia en la sección siguiente. Pero primero, consideraremos un ejemplo un poquito más complicado de un circuito asíncrono en modo fundamental.

EJEMPLO 1

Un circuito en modo fundamental tendrá dos entradas y una sola salida, la cual se convierte en 1 sólo después de la ocurrencia de la última en la secuencia siguiente de combinaciones de entrada; en otro caso $z = 0$:

$$x_1x_2: 00 \ 01 \ 11 \ 10$$

El objetivo es construir una tabla de flujo primitiva.

Cuando el símbolo de entrada es 10, resultarán salidas diferentes dependiendo de la secuencia de entrada precedente. Por consiguiente, esperaríamos dos estados estables diferentes en la columna 10; ¿pero qué ocurre con la columna 11? Incluso si la salida es 0 independientemente de la secuencia de entrada que lleva a ella, la salida que ocurre cuando la entrada 10 sigue a 11 será diferente si la entrada 11 es precedida por 00 01 que si es precedida por 10. En consecuencia, debemos esperar también dos estados estables distintos en la columna 11. Mediante igual razonamiento, lo mismo es cierto para la columna 01. (Razone esto usted mismo.) Sólo en el caso de la entrada 00 resulta irrelevante cómo se alcanza ésta. De tal manera, incluso antes de construirla, esperamos un total de siete estados distintos en la tabla de flujo primitiva.

Marquemos como (a) el estado estable inicial en el cual la entrada es 00. Si indicamos como (b), (c) y (d), respectivamente, los estados subsecuentes que resultan de la secuencia de entrada 00, 01, 11, 10, entonces la tabla de flujo primitiva parcial adquiere la forma que se muestra en la figura 5a.

A continuación volvemos al estado (a). Si la entrada pasa de 00 a 10, el estado estable final no puede ser (d) puesto que la salida no debe ser $z = 1$ con esa secuencia de entrada; denominémosla (e) como en la figura 5b. Luego, empezando desde el estado estable (d), suponga que la entrada va a 11. El siguiente estado estable no debe ser (c) debido a que en ese caso un cambio de entrada que regresara a 11 daría lugar de nuevo al estado (d), con una salida resultante de $z = 1$; pero ésta no es la secuencia de entrada correcta para producir una salida de 1. Por consiguiente, empezando desde (d) con una entrada de 11, deje que el estado alcanzado sea (g) (salida 0). El resto de la tabla de flujo, mostrada en la figura 5b, puede completarse utilizando argumentos similares. (Verifíquelo.) Como se esperaba, hay siete estados. ■

Ejercicio 2. Confirme los restantes dos renglones de la tabla de la figura 5b. ♦

Asignación de salidas para estados inestables

En las tablas de flujo primitivas mostradas en las figuras 4 y 5, no hay valores de salida asociados con los estados inestables. Deben considerarse dos casos. Una transición entre dos estados estables: (a) con las mismas salidas o (b) con salidas diferentes. Como ejemplo del primer tipo, considere la transición de (a) a (b) en la figura 5, donde ambas salidas son 0. Cuando el circuito se encuentra en el estado inestable (b), ¿tendría sentido que la salida pasara a 1, incluso momentáneamente, en el camino del estado (a) al estado (b), ambos con salida 0? La pregunta hace evidente la respuesta:

PS	Estado, z x_1x_2			
	00	01	11	10
a	a, 0	b, 0	-	c, 0
b	a, 0	b, 0	c, 0	
c	-	f, 0	Ⓒ, 0	d, -
d	a, -		g, -	Ⓓ, 1
e	a, 0		g, 0	c, 1
f	a, 0	Ⓓ, 0	g, 0	-
g	-	f, 0	Ⓔ, 0	e, 0

Figura 6. Tabla de flujo para el ejemplo 1.

Cuando ocurre una transición entre dos estados estables que tienen la misma salida, debe asignarse esa misma salida al estado inestable intermedio.

En dos casos de la figura 5 hay una transición entre dos estados que tienen salidas diferentes: de Ⓒ a Ⓓ la salida pasará de 0 a 1, y de Ⓓ a Ⓔ (también de Ⓓ a Ⓐ la salida irá de 1 a 0). Puesto que ocurrirá un cambio en la salida en cualquier caso, la asignación de una salida al estado estable intermedio sólo afectará la rapidez con la que ocurre el cambio de salida.

Ejercicio 3. Decida cómo asignar el valor de la salida a un estado inestable en el párrafo precedente si hay un requerimiento de diseño en cuanto a que el cambio en la salida ocurre (a) lo más rápido posible y (b) lo más lento posible. ¿Qué haría usted si no hubiera especificaciones? ♦

La tabla de flujo completa para el ejemplo 1 se muestra en la figura 6. Las salidas no se especifican cuando una transición ocurre entre estados estables con salidas diferentes.

3 REDUCCIÓN DE MÁQUINAS INCOMPLETAMENTE ESPECIFICADAS

El hecho de que no estén especificados los estados y salidas en algunas columnas de la tabla de flujo no constituye una diferencia básica entre las máquinas asíncronas en modo fundamental y las máquinas secuenciales síncronas. Las últimas también pueden tener los estados o las salidas siguientes sin especificar. Por esta causa, el tema de esta sección se aplica de igual modo tanto a las tablas de flujo de máquinas asíncronas como a las tablas de estados de máquinas síncronas incompletamente especificadas.

Un concepto importante que se usó en la reducción de tablas de estados (completamente especificados) en el capítulo 6 fue la equivalencia. (Consulte su significado si lo requiere.) Cuando hay estados y salidas siguientes sin especificar en una tabla de estados o tabla de flujo, esta definición carece de sentido. Como resultado de alguna entrada, si se presentara una transición hacia un estado siguiente no especificado, ¿qué sucedería al arribar la entrada siguiente? Puesto que se desconoce el estado de la máquina cuando arriba la entrada, no sabremos cuál será el estado o salida siguientes. Así, la equivalencia resulta imposible en casos de este tipo.

La longitud de una secuencia de entrada debe, por tanto, estar limitada; si una combinación de entrada particular envía a la máquina a un estado siguiente que no está especificado, no se permiten entradas adicionales. Se dice que una secuencia de entrada será *aplicable a un estado* si, estando la máquina en ese estado, no la lleva un estado siguiente no especificado, excepto posiblemente en la última entrada. En el caso asíncrono, se impuso la restricción de que ningún par de bits de entrada cambiarán de valor de modo simultáneo. Esto de manera automática hace que resulten aplicables todas las secuencias de entrada permitidas. De esta forma, en la tabla de flu-

jo de la figura 6, suponga que el estado estable es f en la entrada 01. El único estado siguiente no especificado ocurre para la entrada 10. Pero esta entrada no es aplicable. Para cualquier otra entrada, no se encontrarán estados siguientes no especificados. En consecuencia, será aplicable de manera automática la condición de que para cualquier secuencia de entrada ningún par de bits de entrada cambiarán en forma simultánea.

Regresemos ahora a las salidas no especificadas. Suponga que se aplica una secuencia de entrada a una máquina que empieza en cualquiera de dos estados iniciales. En cualquier combinación de entrada particular, ya sea que las salidas de los dos estados estén ambas especificadas, o que sólo se especifique una, o que no se especifique ninguna. Cuando ambas salidas no se especifican, decimos que ellas *no provocan conflicto* entre sí. Ahora bien, además de esto, suponga que cuando ambas salidas se especifican, éstas son las mismas. Esto es causa de júbilo, y lleva a la siguiente definición:

Suponga que para cualquier secuencia de entrada aplicada a cada uno de dos estados iniciales en una máquina, las secuencias de salida producidas no entran en conflicto. Los dos estados se definen como compatibles.

De este modo, la compatibilidad en máquinas incompletamente especificadas es la contraparte a la equivalencia en máquinas completamente especificadas.

La idea de compatibilidad puede extenderse a más de dos estados. De esa manera, un conjunto de estados recibe el nombre de *conjunto compatible* si todos los miembros del conjunto son compatibles entre sí. En el conjunto de tres estados A , B y C , suponga que los pares AB , BC y AC son todos compatibles. Entonces el conjunto ABC es compatible. Por tanto, para determinar la compatibilidad de conjuntos más grandes que pares, sólo necesitamos examinar todos los pares compatibles. (¿Qué es lo que tendría que examinarse si se desca verificar la compatibilidad de conjuntos mayores que tripletas?)

La aplicación de estos conceptos permite la reducción tanto de tablas de estados incompletamente especificados para máquinas secuenciales síncronas como tablas de flujo para máquinas asíncronas. Ésta es la tarea a la que nos abocaremos a continuación.

La tabla de fusión

Los conceptos planteados en esta sección se introducirán en términos de la tabla de flujo de la figura 6. La compatibilidad de dos estados se definió en términos de las secuencias de salida que resultan de una sucesión de entrada. Para cada bit de entrada aplicado a cada uno de dos estados, existe un par de estados siguientes.⁵

A los dos estados siguientes que resultan de la aplicación de una entrada específica a cada uno de los dos estados presentes, se les llama par implicado de estados.

De tal manera, para el par presente de estados bf en la figura 6, el par implicado para la entrada 11 es cg . En ocasiones el "par" implicado no es un par, debido a que los dos estados siguientes son los mismos o en virtud de que uno o ambos de ellos no están especificados. En tales casos, no se lista el "par implicado".

Compatibilidad

En el proceso de decidir la equivalencia (indistinguibilidad) de dos estados para circuitos secuenciales síncronos completamente especificados, encontramos primero una condición para

⁵ Para simplificar, utilizaremos los conceptos "aplicación de una entrada a un estado" cuando queramos decir "aplicar una entrada a una máquina cuando esta última se encuentra en ese estado".

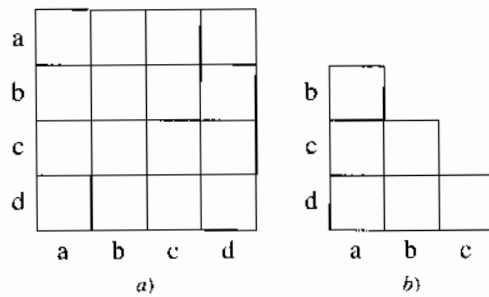


Figura 7. Concepto de la tabla de fusión.

que los dos estados sean distinguibles. Luego establecimos un procedimiento de manera que en cada paso un conjunto de estados no era distinguible. Para el problema presente realizaremos algo análogo: buscaremos una manera de decidir si dos estados no son *incompatibles*. Esto nos llevará a la larga a una conclusión acerca de la compatibilidad.

Es fácil ver que si dos estados S_1 y S_2 son compatibles, entonces los pares implicados para todos los símbolos de entrada son compatibles. Si no, ello significaría que, *empezando a partir del par implicado*, las secuencias de salida no serían las mismas cuando ambas están especificadas. Sin embargo, aquellas secuencias de salida que empiezan desde el par implicado constituyen al menos el primer símbolo de las secuencias de salida desde S_1 y S_2 ; y sabíamos que aquellas secuencias de salidas eran las mismas cuando fueron especificadas, puesto que los dos estados se especificaban como compatibles.

A partir de esto, se desprende también que si un par implicado por el par S_1 S_2 no es un par compatible, entonces S_1 S_2 no es un par compatible. Esto lleva a la siguiente conclusión:

En una máquina incompletamente especificada, dos estados S_1 y S_2 no son incompatibles si y sólo si (a) para cada entrada sus salidas no entran en conflicto y (b) sus pares implicados no son incompatibles.

Este resultado constituye la base de un procedimiento para determinar todos los pares compatibles —y, a partir de ellos, todos los grupos más grandes de compatibles.

Construcción de la tabla de fusión

Necesitamos un método sistemático para exhibir *todos* los pares de estados en una máquina. Uno consistiría en crear una matriz que listara todos los estados vertical y también horizontalmente. Un arreglo de este tipo se muestra en la figura 7a. Cada celda en la tabla representa un par de estados; sobre la diagonal principal, el "par" corresponde precisamente a un estado. Puesto que el concepto que nos ocupa es la compatibilidad (o incompatibilidad), no necesitamos las celdas ab ni las ba, por ejemplo. Esto es, si a es compatible con b, entonces b es compatible con a. De tal modo pueden omitirse los cuadrados ya sea arriba de la diagonal dibujados desde la vertical hasta la horizontal d, o debajo de esa diagonal. Además, las celdas exactamente sobre la diagonal (correspondientes a aa, bb, etc.) también son innecesarias, ya que cualquier estado es compatible consigo mismo. Así, puede dibujarse una tabla modificada (de escalera), como se indica en la figura 7b.

La celda superior correspondiente a aa se ha eliminado, de modo que el listado vertical de estados empieza a partir de b. En forma similar, sobre la parte inferior derecha, ya no aparece la celda que corresponde a dd, por lo que el listado de estados empieza en a pero termina justamente antes del último estado. De ese modo la tabla toma la forma de una escalera triangular: se elimina la diagonal principal y todo lo que está arriba de ella en la matriz original.

Para ilustrar el uso de una tabla de fusión, empezamos con la tabla de flujo en la figura 6. El primer paso es trivial y consiste en establecer la estructura de la tabla amalgamada. La tabla

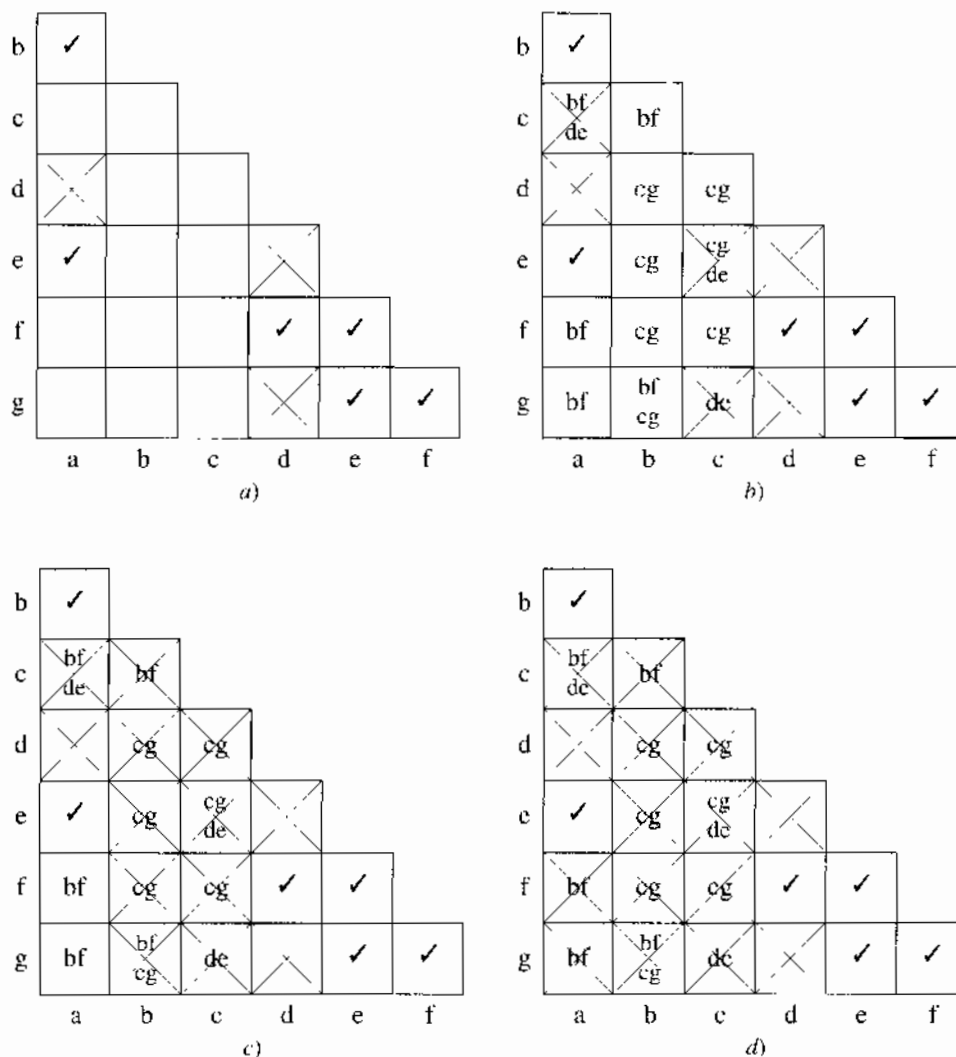


Figura 8. Terminación de la tabla de fusión.

de flujo de la figura 6 tiene siete estados; por ello la tabla amalgamada contará con seis renglones y seis columnas. (Integre esta tabla en blanco usted mismo y efectúe cada paso según se vaya indicando.) Utilizando la tabla de flujo, la cuestión más sencilla por hacer a continuación es determinar aquellos estados que son incompatibles en virtud de que tienen salidas diferentes para la misma entrada. Por ejemplo, para la entrada 10 las salidas relativas a los estados a y d son diferentes en la figura 6. De tal modo, la celda correspondiente en la figura 8a está cruzada. De acuerdo con la tabla de flujo de la figura 6, verifique las otras dos celdas que están cruzadas en la figura 8a.

A continuación examinamos cada par de estados cuya celda correspondiente no se ha cruzado aún. Para cada par de estados que no son incompatibles, lo que intentamos hacer primero en la figura 6 es determinar aquellos cuyo siguiente "par" de estados consiste en:

- Un solo estado, debido a que cada estado siguiente es el mismo o uno de ellos no está especificado;
- Ningún estado en lo absoluto, en virtud de que ambos estados no están especificados.

En cada uno de estos casos el par correspondiente de estados es compatible, de manera que anotamos un \checkmark (marca de verificación) en la celda correspondiente. El resultado hasta este punto se muestra en la figura 8a; verifique que éstos son correctos y que no hay otros.

Hasta el momento, entonces, hemos tratado con aquellos pares de estados que son en definitiva incompatibles y con aquellos que son definitivamente compatibles. Para cada par de estados restantes, el siguiente paso utilizando una tabla de flujo consiste en determinar los pares implicados de estados siguientes; pares reales. Por ejemplo, para la entrada 01 y el par de estados inicial ac, el par implicado en la figura 6 es bf. Escribimos bf en la celda ac de la tabla amalgamada. Prosiga con este esquema para todos los pares de estados restantes en la tabla de flujo. Luego confirme su trabajo utilizando la figura 8b.

En realidad, le hicimos trampa al efectuar un paso adicional. Advierta que la figura 8b tiene tres celdas cruzadas más que la figura 8a. Esto resulta de un examen de los *pares implicados* dentro de cada celda en la figura 8b para determinar si algunos de ellos son pares que sean incompatibles de acuerdo con la figura 8a. Por ejemplo, un par implicado en la celda ac es de. Pero de la figura 8a sabemos que los estados d y e son incompatibles. Por ello, puesto que el par ac implica al par de, y de es un par incompatible, entonces ac también debe ser incompatible. Ésta es la razón por la que cruzamos el cuadrado ac. Verifique las demás celdas cruzadas en la figura 8b.

El proceso se continúa ahora observando si cada par que acaba de descubrirse incompatible (ac, ce y cg) es un par implicado escrito en cualquier celda que no se ha cruzado aún. Los pares incompatibles ac y ce, por ejemplo, no son pares implicados escritos en cualquier celda. Pero cg es un par implicado que aparece en seis celdas diferentes; por consiguiente, puesto que cg es un par incompatible, la totalidad de estas seis celdas deben estar ahora cruzadas, como se muestra en la figura 8c; los pares correspondientes son incompatibles.

Después de que éstas se han cruzado, advertimos que el único par implicado que queda en cualquier celda no cruzada es bf; y el par bf es incompatible en virtud de que su celda correspondiente se ha cruzado en la figura 8c. Así, se han cruzado ahora las dos celdas que contienen a bf como un par implicado en la figura 8c. El resultado final se muestra en la figura 8d.

Determinación de coberturas cerradas mínimas

Aunque el proceso que acaba de presentarse parece lento, lo que es lento es *describirlo*, no realizarlo. En realidad, el proceso es algorítmico; por consiguiente, puede hacerse un programa computacional (y así se ha hecho) para llevarlo a cabo.⁶ El objetivo consiste en descubrir todos los pares de estados que son compatibles, en virtud de que no se han cruzado. En este ejemplo, los pares compatibles son:

$$\{ab, ae, df, ef, eg, fg\} \rightarrow \{ab, ac, df, efg\}$$

Advierta que los tres estados e, f y g son compatibles en pares; de tal modo, la tripleta efg es un conjunto de estados compatible. Ningún otro par puede combinarse para formar una tripleta compatible; tampoco es posible combinar cualquiera otros con efg para formar un conjunto compatible más grande. Estas observaciones conducen a algunas definiciones:

Un conjunto compatible de estados C_1 cubre otro conjunto compatible de estados C_2 si cada estado en C_2 está contenido en C_1 .

Un conjunto compatible de estados es compatible máximo si no está cubierto por cualquier otro conjunto compatible de estados.⁷

⁶ Tres herramientas de asignación de estados (Jedi, Nova y Mustang) distribuidas con las herramientas de diseño de circuitos integrados Octtools contienen implementaciones de este procedimiento.

⁷ Cuando no hay posibilidad de confusión, se omitirá "de estados" "Cobertura mínima cerrada" constituye la terminología estándar.

	S x_1x_2				Z x_1x_2			
	00	01	11	10	00	01	11	10
ab \rightarrow 1	①	①	2	4	0	0	0	0
c \rightarrow 2	—	4	②	3	—	0	0	—
d \rightarrow 3	1	—	4	③	0	—	0	1
efg \rightarrow 4	1	④	④	④	0	0	0	0

Figura 9. Tabla de flujo reducida.

Con esta terminología, advertimos que cada uno de los anteriores pares compatibles cubre a cada uno de los estados que integran el par. Los pares ab, ae y df son máximos, aunque ef, eg y fg no lo son puesto que todos ellos son cubiertos por efg. Puesto que el estado c no es cubierto por alguno de los pares compatibles, es un máximo. Advertimos que los estados individuales en la tabla de flujo original se han *amalgamado* en combinaciones de estados; de tal modo, la tabla en la figura 8 recibe el nombre de *tabla amalgamada*.

Es evidente una diferencia crucial entre conjuntos de estados compatibles y conjuntos de estados equivalentes en una máquina síncrona: los máximos en la lista anterior no son disjuntos; se traslapan. Los estados a, e y f aparecen cada uno en dos compatibles diferentes.

La tabla amalgamada es una herramienta y mediante su uso es posible obtener el conjunto de todos los compatibles máximos para una tabla de flujo determinada. Sin embargo, el principal problema de la minimización de máquina (encontrar otra máquina con el menor número de estados cuyo desempeño sea indistinguible del correspondiente a la máquina original) sigue sin resolverse en forma completa. Si se formará una nueva tabla de estados mediante un conjunto de compatibles, éstos deben cubrir todo estado original. Además, el "estado siguiente" que resulta de cualquier entrada debe cubrirse mediante uno de los compatibles presentes. Lo anterior nos lleva a la siguiente idea:

Un conjunto de estados compatibles es cerrado si un estado compatible implicado por cualquiera de ellos es cubierto por algún estado compatible en el conjunto.

Estamos listos para encontrar una máquina con el menor número de estados cuyo desempeño es indistinguible de la máquina original. Lo que necesitamos es un conjunto mínimo de compatibles que esté cerrado y que cubra todos los estados de la máquina original: una *cobertura mínima cerrada*.

En el ejemplo anterior, encontramos que el conjunto de máximos era

$$\{ab, ae, c, df, efg\}$$

No todos estos máximos se necesitan: el conjunto $\{ab, c, df, efg\}$ constituye una cobertura cerrada mínima, aunque una cobertura mínima cerrada no requiere que los conjuntos de compatibles sean máximos. Advertimos que el estado f es cubierto tanto por df como por efg; podríamos considerar eliminar f de df o de efg. Los siguientes conjuntos de cobertura mínima cerrada se producen si lo anterior se lleva a cabo: $\{ab, c, df, eg\}$ o $\{ab, c, d, efg\}$.

Ejercicio 4. Confirme que cada conjunto de estados en el enunciado anterior incluye todos los estados originales, es mínimo y cerrado. ♦

Existen procedimientos para buscar de manera sistemática coberturas mínimas cerradas similares al algoritmo Quine-McCluskey para máquinas completamente especificadas. Sin embargo, no seguiremos este tipo de asuntos en forma general aquí.

$y_1 y_2$	$Y_1 Y_2$ $x_1 x_2$				$y_1 y_2$	$Y_1 Y_2$ $x_1 x_2$				z $x_1 x_2$			
	00	01	11	10		00	01	11	10	00	01	11	10
1 → 00	00	00			00	00	00	01	10				
2 → 01			01		01	—	10	01	11	—			—
3 → 11				11	11	00	—	10	11		—		1
4 → 10		10	10	10	10	00	10	10	10				
a)					b)								

Figura 10. Tablas de transición y salida para la tabla reducida de la figura 9.

Supongamos que el conjunto {ab, c, d, efg} se elige como una cobertura mínima cerrada. La tabla de flujo reducida para esta elección se muestra en la figura 9. (La tabla de salida se ha mostrado de manera separada, por conveniencia.) Es posible realizar varias observaciones. En primer lugar, cuando se combinan un estado inestable y uno estable en un compatible, el estado que resulta debe ser estable.

Segundo, a diferencia de la tabla de flujo primitiva, la tabla reducida puede tener más de un estado estable en cada renglón. Puesto que a cada renglón se le asignarán valores específicos de las variables secundarias, como veremos, todos los estados estables en un renglón determinado tienen el mismo número de secundarias; éstos se distinguen entre sí únicamente por los estados de entrada. Lo anterior constituye una distinción básica de los circuitos secuenciales síncronos, donde las entradas no desempeñan ningún papel en la especificación del estado. Por ejemplo, la única manera de que dos estados estables en el primer renglón de la figura 9 —ambos marcados 1— difieran entre sí, estriba en sus valores de entrada: $x_1 x_2 = 00$ en un caso y 01 en el otro.

Por último, algunas entradas que no se especificaron en una tabla de flujo primitiva se especifican ahora en la tabla reducida. La fusión de ① y ② en la columna 11 requiere la combinación del c inestable con una entrada no especificada. El resultado es el estado inestable 2, como se muestra en la figura 9, renglón 1 y columna 11. Esto no significa que sea ahora posible una transición de ① para $x_1 x_2 = 00$ a ② para $x_1 x_2 = 11$. Se sigue aplicando la suposición de que únicamente un bit de entrada cambia en cualquier tiempo. Una transición permitida va de ① en la columna 01 a ② en la columna 11. De manera similar, la transición de ① a ④ va de la columna 00 a 10, y no de la columna 01 a la 10.

Tablas de transición

Una vez que se dispone de una tabla de flujo reducido, debe realizarse una asignación de valores de variables secundarias. En los circuitos asíncronos en modo fundamental, surgen algunas dificultades importantes al intentar asignar valores secundarios. Estas dificultades se descartarán temporalmente aquí; se analizará en secciones subsiguientes.

La tabla de cuatro estados en la figura 9 requiere dos variables secundarias para su implementación. Establezcamos de manera arbitraria la asignación $y_1 y_2 = 00, 01, 11, 10$ para los estados estables en orden. La tabla de transición parcial resultante que incluye sólo las entradas de estados estables se presenta en la figura 10a. Advierta que las entradas en cualquier renglón son las excitaciones que corresponden a esa combinación de variables secundarias. Para estados estables, las excitaciones tienen los mismos valores que las secundarias como se confirma en la tabla parcial. La pregunta en este caso es, ¿qué valor de excitación deben tomar los estados inestables? Puesto que, luego de cierto retardo, los estados inestables se convertirán en los estados estables correspondientes, el valor de excitación para un estado inestable debe ser el mismo que los valores secundarios del estado estable correspondiente. Utilizando este concepto, las

1. Construya una tabla de flujo primitiva que cumpla las especificaciones del problema —un estado estable por renglón con salidas especificadas sólo para estados estables. Especifique las salidas de los estados inestables de manera que no ocurran parpadeos de salida momentáneos y se satisfagan posibles cambios rápidos o lentos de la salida en las especificaciones.
2. Obtenga una tabla reducida mínima utilizando una tabla de fusión para determinar compatibilidades.
3. Asigne valores secundarios a los estados.
4. A partir de la tabla mínima y de la asignación de valor secundario, construya tablas de transición y salida. Obtenga expresiones en éstas para las salidas y las excitaciones.
5. Implemente estas expresiones en un diagrama de circuito.

Figura 11. Procedimiento de diseño de circuito en modo fundamental.

tablas terminadas de transición y de salida que se producen serán como se indica en la figura 10b. Confirme todas las entradas en cada tabla.

Lo que acaba de llevarse a cabo mediante un ejemplo específico en la explicación anterior ilustra un procedimiento de diseño general que se resume en la figura 11.

EJEMPLO 2

Un circuito secuencial síncrono tendrá dos entradas y una salida. Ésta se va a convertir en 1 (si no es ya 1) cuando la entrada cambie de $x_1x_2 = 00$ a 10. Se volverá 0 (si no es ya 0) cuando la entrada cambie de 00 a 01.

De acuerdo con el procedimiento de diseño, el primer paso consiste en construir la tabla de flujo primitiva. Observamos que, luego de que la salida ha tomado el valor 1, se mantendrá en 1 hasta que la entrada alcance de nuevo 00 y después 01. De manera similar, cuando $z = 0$, permanecerá en 0 hasta que la entrada arribe a 00 y luego a 10. Por consiguiente, podríamos esperar una salida tanto de 0 como de 1 para cada símbolo de entrada, dependiendo de la terminación de la secuencia de entrada de ese símbolo particular. Eso equivale a un total de ocho estados estables, dos estados estables distintos para cada símbolo de entrada.

Iniciando en el estado \textcircled{a} con el símbolo de entrada $x_1x_2 = 00$, la tabla de flujo primitiva toma la forma que se muestra en la figura 12a. (Verifique cada uno de los renglones.) Como se esperaba, hay ocho estados estables, dos escritos en cada columna. El siguiente paso es construir la tabla amalgamada, lo cual se efectúa en la figura 12b. (Compruebe los detalles.)

A partir de la tabla de fusión se determinan los pares compatibles y, con base en éstos, las compatibilidades máximas:

$$\text{conjunto de compatibles máximos} = \{ab, bed, efg, eh\}$$

Hay dos estados redundantes, b y e. Puesto que los conjuntos compatibles implicados por los compatibles máximos no contienen pares o conjuntos mayores, cada estado redundante puede omitirse de cualquier conjunto compatible en el que aparezca, o es posible retener cada estado en ambos compatibles. Las consideraciones que guían esta elección se explicarán en una sección posterior. Por ahora, omitimos de manera arbitraria el estado b del compatible ab y el estado e del compatible eh. La tabla reducida mínima se construye como se indica en la figura 13a.

El siguiente paso radica en asignar valores secundarios a los estados. (Como se señaló antes, los principales problemas conectados con este paso se expondrán en las secciones finales.) Por ahora, se realiza la asignación que se indica en la figura 13b, dando lugar a la tabla de transición que se muestra.

PS	Estado x_1x_2			
	00	01	11	10
a	(a), 0	b	-	e
b	a	(b), 0	c	-
c	-	b	(c), 0	d
d	a	-	c	(d), 0
e	h	-	f	(e), 1
f	-	g	(f), 1	e
g	h	(g), 1	f	-
h	(h), 1	b	-	e

Máximos: ab, bcd, cfg, eh

a)

b	✓						
c	de	✓					
d	de	✓	✓				
e	ah	ah	cf	de			
f	bg	bg	✓	cf	de	✓	
g	ah	bg	bg	ah	cf	✓	✓
h		ah	de	ah	de	✓	bg

b)

Figura 12. Tabla de flujo primitiva y tabla de fusión para el ejemplo 2.

PS	Estado x_1x_2				z x_1x_2				y_1y_2 x_1x_2					
	00	01	11	10	00	01	11	10	y_1y_2	00	01	11	10	
a → 1	(1)	2	—	3	0	×	×	×	1 → 00	(00)	01	—	10	
bcd → 2	1	(2)	(2)	(2)	×	0	0	0	2 → 01	00	(01)	(01)	(01)	
efg → 3	4	(3)	(3)	(3)	×	1	1	1	4 → 11	(11)	01	—	10	
h → 4	(4)	2	—	3	1	×	×	×	3 → 10	11	(10)	(10)	(10)	
	a)								b)					

a)

b)

Figura 13. Tabla de flujo mínima y tabla de transición del ejemplo.

Advierta que el estado ②, al que se asigna el valor secundario 01, aparece tres veces en el renglón 2. Los estados *totales* representados por estos tres son todos distintos, distinguiéndose uno del otro por sus valores de entrada. A pesar de eso, puesto que éstos tienen los mismos valores secundarios, se les da el mismo nombre. Lo mismo es cierto para el estado estable ③ en el último renglón.

En una tabla de flujo *primitiva*, la única manera para llegar a un estado estable es por medio de una transición vertical a través de un estado inestable. En una tabla *reducida*, en cambio, es posible alcanzar (horizontalmente) un estado estable sin otra cosa que cambiar un estado de entrada individual. Así, en la figura 13a, el estado ② en la columna 01 puede alcanzarse a partir del estado ① en la columna 00. Es factible también que se alcance de manera horizontal desde el estado ② en la columna 11. Sin embargo, el estado ② en la columna 10 se alcanza sólo con un cambio de valores de entrada del estado ② en la columna 11, sin un cambio de valores secundarios.

Como último paso, las expresiones para las funciones de excitación se obtienen de modo directo de la tabla de transición, o de los mapas de excitación correspondientes a funciones de excitación individuales obtenidas de esta tabla. Los mapas para las excitaciones y la salida que se

	x_1x_2			
	00	01	11	10
y_1y_2	00		×	1
	01			
	11	1	×	1
	10	1	1	1

$$Y_1 = y_1y_2' + x_2'y_1 + x_1y_2'$$

	x_1x_2			
	00	01	11	10
y_1y_2	00		1	×
	01		1	1
	11	1	1	×
	10	1		

$$Y_2 = x_1y_1'y_2 + x_1'x_2'y_1 + x_2y_1' + x_2y_2$$

	x_1x_2			
	00	01	11	10
y_1y_2	00		×	×
	01	×		
	11	1	×	×
	10	×	1	1

$$z = y_1$$

Figura 14. Funciones de excitación y salida para el ejemplo 2.

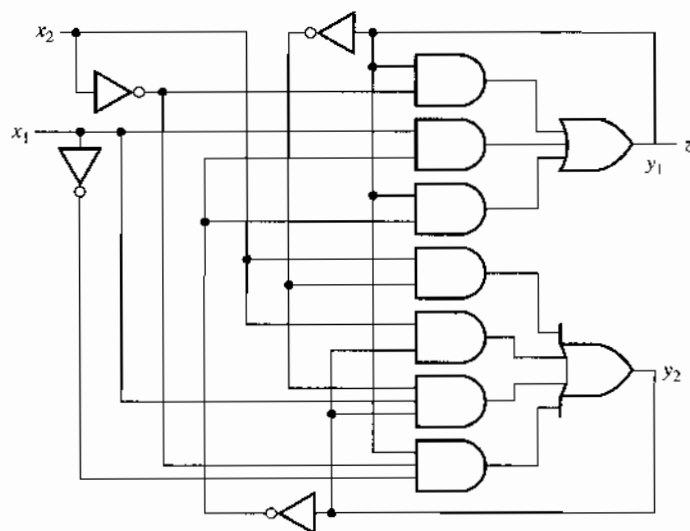


Figura 15. Implementación del circuito del ejemplo 2.

obtienen de la figura 13b se presentan en la figura 14. Las expresiones que se producen para las excitaciones y la salida se dan en los mapas. (Confirme todas éstas.)

El circuito que implementa estas funciones se muestra en la figura 15. (Compruebe cada una de las afirmaciones anteriores.) ■

4 CARRERAS Y CICLOS

Los circuitos en modo fundamental se definieron como circuitos asíncronos sensibles a cambios de nivel en las entradas donde sólo ocurre un cambio de entrada primaria a la vez. La causa de esta restricción tiene que ver con la dificultad práctica de asegurar cambios absolutamente simultáneos. Puesto que no es posible que existan dos cambios exactamente simultáneos si la escala de tiempo se hace lo suficientemente fina, en la práctica únicamente son factibles los cambios no simultáneos. La restricción real en discusión, entonces, es que debe haber suficiente separación en el tiempo entre dos cambios de entrada para asegurar que el circuito haya alcanzado un estado estable como resultado de un cambio antes que se permita la ocurrencia del siguiente cambio de entrada.

y_1y_2	$\begin{matrix} Y_1Y_2 \\ x_1x_2 \end{matrix}$			
	00	01	11	10
1 \rightarrow 00	00	01	—	11
2 \rightarrow 01	00	01	01	01
3 \rightarrow 11	10	11	11	11
4 \rightarrow 10	10	01	—	11

Figura 16. Tabla de transición para la asignación modificada.

Un vistazo al circuito en modo fundamental de la figura 2 indica, a este respecto, que las entradas secundarias no difieren de las primarias. Por razones prácticas, es imposible garantizar tampoco cambios simultáneos para éstas. De tal manera, un cambio secundario debe estar bastante separado en el tiempo de otro cambio secundario de modo que un estado estable se haya alcanzado antes de que otra variable secundaria cambie su valor.

Observe de nuevo la tabla de transición de la figura 13b. Empezando en cualquier estado estable, suponga que ocurre un cambio de entrada permisible. ¿Qué cambios se producen en los valores secundarios en su curso hacia el siguiente estado estable? Partiendo de $x_1x_2y_1y_2 = 0000$, considere que x_1x_2 cambia en 01. Según la tabla, la transición que se requiere es hacia $y_1y_2 = 01$, a través del 01 inestable. Esto es, únicamente y_2 cambia su valor, de 0 a 1. De manera similar, desde el mismo estado estable inicial, suponga que x_1x_2 cambia a 10. La transición secundaria que se requiere es a $y_1y_2 = 10$, a través del 10 inestable. También en este caso, sólo una secundaria cambiará su valor: y_1 de 0 a 1. En esta tabla, todos los demás cambios de entrada permisibles requieren sólo que cambie una variable secundaria. (En otros casos, los cambios de entrada permisibles pueden conducir a que todas las secundarias permanezcan sin cambio, como se ilustra en la sección precedente.)

Carreras críticas y no críticas

Para evitar que dos secundarias cambien sus valores al mismo tiempo, debemos determinar cómo es posible que dos cambios de este tipo ocurran de manera simultánea. Éstos dependen de la asignación de valores secundarios a los estados. Regresemos de nuevo a la figura 13b y supongamos que se intercambian las asignaciones de los estados 3 y 4. El resultado se presenta en la tabla de transición de la figura 16.

Considere ahora el cambio de entrada permisible del estado total $x_1x_2y_1y_2 = 0000$ a $x_1x_2 = 10$. Estudie la tabla con cuidado a medida que avance. La transición requerida es hacia el estado inestable 11, a partir del cual el circuito debe pasar al 11 estable.

Sin embargo, esto requiere un cambio de secundarias de 00 a 11. Dicho cambio *simultáneo* requerido de dos variables secundarias es imposible en la práctica debido a que necesita que los retardos de las entradas para cada variable secundaria sean absolutamente iguales. En la práctica, ya sea y_1 o y_2 cambiarán su valor primero. Si y_1 cambia primero, la transición en la columna 10 pasa de $y_1y_2 = 00$ a 10. Pero el estado al cual el circuito se dirige en la columna 10, renglón 10, permanece en el estado inestable 11, a partir del cual ocurre la transición requerida a $y_1y_2 = 11$. Así, la transición requerida originalmente se alcanza en dos pasos. De $y_1y_2 = 00$ a 10 y a 11, pero se logra.

Suponga ahora que y_2 cambia primero, desde el valor original $y_1y_2 = 00$ a 01. El circuito alcanza entonces el estado estable 01, una transición incorrecta. La máquina está ahora en un estado estable incorrecto y su desempeño subsecuente será incorrecto. El proceso puede considerarse como una carrera entre dos secundarias para ver cuál de ellas puede cambiar primero.

		$Y_1 Y_2$			
		$x_1 x_2$			
$y_1 y_2$		00	01	11	10
00		00	00	01	10
01		—	11 ← 01		11
11		00	10	10	11
10		00	10	10	10

a)

		$x_1 x_2$			
		00	01	11	10
$y_1 y_2$	00			1	
	01	×		1	1
	11		×		1
	10				

b)

		$x_1 x_2$			
		00	01	11	10
$y_1 y_2$	00			1	
	01	×	1	1	1
	11				1
	10				

c)

Figura 17. Formación de un ciclo para evitar una carrera crítica. a) Tabla de transición. b) $Y_{2\text{vieja}}$. c) $Y_{2\text{nueva}}$.

Existe una carrera en un circuito secuencial en modo fundamental si, como consecuencia de un cambio de entrada permisible, se requiere que cambie más de un valor secundario a un tiempo. Si el estado estable final que se alcanza depende del orden en el cual las secundarias cambian sus valores, entonces la carrera es crítica.

En el diseño de un circuito, no puede tolerarse una carrera crítica: debe evitarse *siempre*. Para las asignaciones secundarias de la figura 16, existe una carrera crítica. En el diseño del mismo circuito, encontramos una asignación en la figura 13 que evita esta carrera crítica. La solución resulta clara en este ejemplo: efectuar las asignaciones de valores para los estados de manera que se eviten las carreras críticas. Sin embargo, la situación no siempre es tan sencilla, como lo descubriremos más adelante.

En otros casos es posible una situación diferente de la que acaba de ilustrarse. En la tabla de la figura 10, suponga un estado total estable $x_1 x_2 y_1 y_2 = 10(11)$ cuando la entrada cambia a un $x_1 x_2$ permisible = 00. El requerimiento implica que los valores secundarios vayan a 00, un cambio simultáneo en dos variables secundarias. Se produce una carrera, pero esta vez, sin importar cuál secundaria cambie primero, el estado estable final es $y_1 y_2 = 00$, el valor deseado. Se afirma que la carrera será *no crítica*, y no es necesario perder el sueño por ello. Verifique que el estado estable que se alcanza es 00.

La tabla de transición de la figura 10 en realidad no es *tan* inocente; contiene otra carrera que ocurre del estado 11(11) al deseado 01(10). Si y_2 cambia antes que y_1 , de 1 a 0, el circuito pasará al estado estable 00, que no es un estado correcto. Esta carrera es, en consecuencia, crítica. ¿Qué hacer? Advierta que hay un valor no relevante en el renglón 11, columna 01. Suponga que el valor irrelevante se sustituye por valores secundarios 10 en esa posición. Ahora bien, si pudiéramos dirigir los cambios en los valores secundarios desde el 11 original en la columna 11 hasta el renglón 11, columna 01, entonces seríamos guiados al estado deseado 01 a partir de ahí. Lo anterior puede conseguirse cambiando la asignación de las secundarias en el renglón 01, columna 01, de 10 a 11. La tabla de transición alterada toma la forma que se muestra en la figura 17.

Las flechas muestran la trayectoria que sigue el cambio en el estado total:

$$x_1 x_2 y_1 y_2: 11(11) \rightarrow 0101 \rightarrow 01111 \rightarrow 01(10)$$

En cada paso, sólo ha cambiado una variable (primaria o secundaria), y se alcanza el estado estable final que se desea.

Ciclos y oscilaciones

La idea de que un estado estable pueda alcanzarse desde otro estado estable a través de una secuencia de estados inestables intermedios puede formalizarse mediante la siguiente definición:

Cuando un circuito secuencial en modo fundamental realiza una transición de un estado estable a otro por medio de una secuencia única de estados inestables, afirmamos que existe un ciclo.

Ejercicio 5. De acuerdo con la figura 16, determine una expresión booleana en la forma de suma de productos para la excitación Y_1 . Repita para la Y_1 de la figura 10b, y compare las dos expresiones.

Respuesta⁸

¿Qué sucedería si, empezando en un estado estable, se produce una secuencia de estados inestables sin alcanzar nunca otro estado estable antes de que ocurra la siguiente entrada? Debido a la variabilidad en los retardos de compuerta, el estado final del circuito en el siguiente cambio de entrada será incierto. Por consiguiente, también lo será el desempeño subsecuente. Esto nos lleva a la siguiente idea:

Afirmamos que existe una oscilación si un cambio de entrada permisible inicia una transición desde cierto estado estable a través de una secuencia de estados inestables sin alcanzar nunca otro estado estable antes del siguiente cambio de entrada.

No puede tolerarse una oscilación, y deben tomarse todas las medidas que sean necesarias para evitarlo.

Por ahora alguno de los lectores estará ansioso por determinar qué posible conexión tiene este manejo de las tablas de flujo con la implementación del circuito real. Para responder a la pregunta, determinaremos las implementaciones del circuito respecto a la asignación de la figura 10 y la asignación modificada de la figura 17. Parte de la tarea la realizará usted.

Lo que sigue tratará de manera explícita sólo la excitación Y_2 utilizando tanto la asignación vieja de la figura 10 como la nueva de la figura 17. Las expresiones para la excitación Y_2 son como sigue:

$$Y_{2\text{vieja}} = x_1 x_2' y_2 + x_1 x_2 y_1' \quad (1)$$

$$Y_{2\text{nueva}} = x_1 x_2' y_2 + x_1 x_2 y_1' + y_1' y_2 = Y_{2\text{vieja}} + y_1' y_2 \quad (2)$$

Las implementaciones de estas dos funciones se muestran en la figura 18. La nueva es la misma que la vieja salvo por la compuerta AND agregada.

Analicemos cada uno de estos circuitos para determinar las consecuencias de un cambio en la entrada $x_1 x_2$ de 11 a 01. Para este fin necesitaremos las expresiones para Y_1 , tanto la vieja como la nueva; sucede que éstas son las mismas del ejercicio 5. En cada paso que sigue, deberá confirmar los resultados.

Cuando $x_1 x_2 = 11$, los valores secundarios son $y_1 y_2 = \textcircled{01}$. En este caso x_1 cambiará de 1 a 0. En la figura 18a (el circuito viejo), cuando x_1 cambia de 1 a 0, las salidas de ambas compuertas AND cambian a 0 (después de un retardo de propagación de compuerta).

Después de un retardo de compuerta más a través de la compuerta OR, Y_2 cambia de 1 a 0. En cuanto a Y_1 (cuyo valor es al principio 1), cuando x_1 va de 1 a 0, x_1' pasa de 0 a 1, ocasionando que el tercer término en Y_1 (que representa a una compuerta AND) vaya de 0 a 1. Puesto que

⁸ $Y_1 = x_1 x_2' + x_2 y_1 + x_1' x_2 y_2$. (El último término se convierte en $x_1' y_1' y_2$ si el valor irrelevante se usa para formar un implicante primo.)

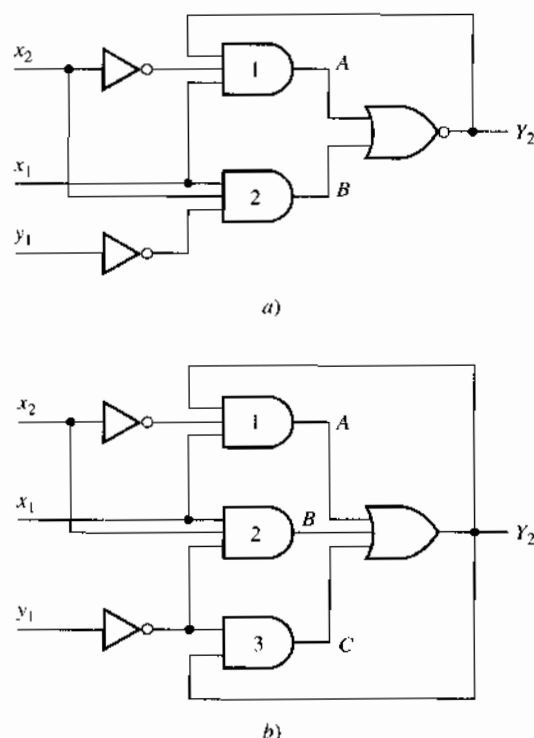


Figura 18. Implementaciones de Y_2 en el ejemplo. a) Asignación vieja. b) Asignación nueva.

al menos una entrada para la compuerta OR de salida en la implementación de Y_1 es 1, el valor de Y_1 conmuta de 0 a 1. Pero este cambio ocurre luego de tres retardos de compuerta; si no se tiene cuidado especial para poner compuertas rápidas en la implementación de Y_1 , el retardo a lo largo de esta trayectoria será un poco mayor que el relativo a la trayectoria de Y_2 . Por consiguiente, y_2 cambiará primero, y por ello $y_1 y_2$ irá de $\textcircled{01}$ a $\textcircled{00}$; ¡la transición incorrecta!

Considere ahora el nuevo circuito para Y_2 en la figura 18b; éste difiere del viejo al tener una compuerta AND adicional. (Recuerde que Y_1 es la misma.) La Y_2 vieja era 1 para $x_1 x_2 = 11$ en virtud de que la salida de la compuerta AND 2 era 1. En el nuevo circuito, las salidas de ambas compuertas AND 2 y 3 son en un principio 1. Cuando la entrada x_1 cambia en 0, la salida de la compuerta AND 2 se convierte en 0, pero la salida de una compuerta AND 3 permanece en 1 hasta que y_1 cambia su valor. Así, y_1 cambia antes que y_2 , y por ello la transición de estado es de $y_1 y_2 = \textcircled{01}$ a $Y_1 Y_2 = 11$, luego de lo cual cambia Y_2 , haciendo que $Y_1 Y_2 = 10$ y, por último, $y_1 y_2 = \textcircled{10}$. De este modo usted observa que las manipulaciones en la tabla de transición tienen efectos en el mundo real.

Con la intención de resumir, advertimos que, de una u otra manera, es necesario evitar las carreras críticas en una tabla de transición. Una forma consiste simplemente en efectuar una asignación apropiada de los valores secundarios: éste fue el caso de la figura 13. Otra forma es triba en utilizar una entrada no especificada para formar un ciclo en una columna en la cual existe una carrera crítica, como en la figura 17. Ninguna de estas posibilidades constituye un remedio universal debido a que

- Quizá no existan entradas sin especificar con cuáles formar un ciclo, y
- Es posible que no haya asignación que evite una carrera crítica —sin medidas adicionales.

Vale la pena identificar una asignación que evita carreras críticas y oscilaciones:

Una asignación válida es cualquier asignación de valores secundarios para la cual no hay carreras críticas ni oscilaciones.

Un problema principal en el diseño del circuito en el modo fundamental radica en encontrar una asignación válida. Una asignación es realmente válida si los estados estables entre los cuales ocurrirá la transición son asignaciones adyacentes determinadas. Sin embargo, como se señaló antes, también será válida una asignación si se forman ciclos de un estado estable a otro. Existen otros mecanismos para determinar asignaciones válidas; uno consiste en dar asignaciones múltiples a ciertos estados de manera que puedan alcanzarse todas las adyacencias requeridas. Esto podría significar que el número de variables de estado —y por ello la complejidad del circuito resultante— se incrementan. Se han descrito métodos en otros trabajos que producen asignaciones *universales* para tablas de flujo con ciertos números de estados, en ocasiones sin requerir variables de estado adicionales. No continuaremos el problema de determinar asignaciones válidas aquí.

5 RIESGOS

El proceso de diseño descrito en las secciones anteriores prosigue a lo largo de una serie de pasos para llegar a una tabla de transición, como la de la figura 17a. Pero la tabla de transición es simplemente otra forma de la tabla de verdad para las funciones de excitación. La única diferencia entre el circuito en modo fundamental y el combinatorio es la retroalimentación en el primero. No hay diferencia en el *proceso* de obtener expresiones adecuadas para las excitaciones a partir de la tabla de transición, por un lado, y en la obtención de una expresión para una función combinatoria partiendo de un mapa lógico o lista de minitérminos, por el otro.

Existe, sin embargo, una diferencia en el *significado*. En el caso de la tabla de transición, algunas de las variables no son entradas independientes sino secundarias. En consecuencia, el problema que se expondrá en esta sección se aplica a los circuitos combinatorios, así como a los circuitos secuenciales en modo fundamental. Las consecuencias, no obstante, son diferentes.

Riesgos estáticos

En la explicación anterior de los orígenes de una carrera en un circuito secuencial en modo fundamental, descubrimos que el retardo de la propagación a través de las compuertas desempeña un papel protagónico. El mismo factor —el retardo— desempeña también otra función, como explicaremos ahora.

Supóngase que nos encontramos en medio de un proceso de diseño en modo fundamental y que llegamos al mapa de una función de excitación Y_1 que se indica en la figura 19. Se indica también la expresión mínima de suma de productos y su realización de circuito. No se ilustra la implementación para la excitación Y_2 , aunque suponemos que los cambios en Y_2 ocurren de manera más lenta que los correspondientes en Y_1 . Supóngase ahora que $y_1 y_2 = \textcircled{11}$ y que $x_1 x_2$ cambia de 11 a 10. De acuerdo con el *mapa*, el valor de Y_1 permanecerá invariable ($Y_1 = 1$); pero considere lo que puede suceder en el *circuito*.

Antes del cambio en x_2 de 1 a 0, las salidas de la compuerta AND fueron $A = 1$ y $B = 0$, por lo que Y_1 fue 1. Después de esto x_2 se convierte en 0; si la compuerta AND 1 es más rápida que la trayectoria a través del inversor y la compuerta AND 2 —lo cual no es improbable— entonces A se vuelve 0 *antes* que B haya cambiado de 0 a 1. Por consiguiente, ambas entradas para la compuerta OR son 0 durante algún breve intervalo de tiempo, provocando que Y_1 se torne 0. Pero Y_1 se retroalimenta a las entradas de ambas compuertas AND. El resultado es que, dependiendo de

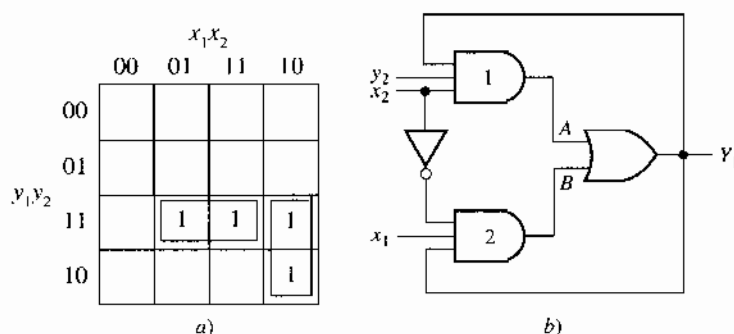


Figura 19. Función de excitación y su implementación de circuito. a) $Y_1 = x_1x_2'y_1 + x_2y_1y_2$. b) Implementación del circuito.

las diferencias en el retardo a lo largo de diversas trayectorias, un cambio momentáneo en Y_1 a 0 puede volverse un cambio permanente, el cual dará lugar a un valor incorrecto de una variable secundaria.

El mismo problema de diferencias en el retardo a lo largo de diferentes trayectorias existiría incluso si éste fuera un circuito combinatorio. En ese caso, sin embargo, un valor incorrecto momentáneo no es un problema, puesto que se corregirá por sí solo. Ésta es la retroalimentación que provoca el problema en este caso.

Ahora bien, generalicemos la idea descrita en el ejemplo anterior. Considere que una tabla de transición de una máquina en modo fundamental requiere que el valor de una o más variables secundarias permanezcan invariables (en 0 o 1) cuando ocurra un cambio de entrada permisible (sólo una variable). Establecemos, entonces, la siguiente definición:

Suponga que el valor de una o más variables secundarias en la tabla de transición de un circuito secuencial asíncrono permanecerá sin cambio (en 0 o 1). Considere además que, como consecuencia de retardos desiguales a lo largo de diferentes trayectorias, es posible que una o más variables secundarias presenten —un cambio de valor momentáneo— debido a un cambio permisible de la entrada. Afirmamos entonces que existe un riesgo estático en el circuito.

(El término *estático* se refiere al hecho de que se supone que todas las secundarias se mantienen sin cambio, esto es, estáticas.)

La existencia de un riesgo estático significa que el circuito es imperfecto y que el problema tendrá que corregirse. Tanto la causa del problema como su remedio son evidentes a partir del mapa lógico de la figura 19a. La expresión para Y_1 se obtiene de los dos implicantes primos que se muestran en el mapa. Cada implicante primo se implementa mediante una compuerta AND. El riesgo ocurre debido a que un cambio en una entrada primaria (x) mueve al circuito de un implicante primo a otro. Durante un corto intervalo de tiempo, es posible que ninguno de los implicantes primos sea operativo. En términos de las compuertas AND, una tenía una salida de 1 y la otra una de 0. En el cambio de la entrada primaria, se supone que ambas compuertas cambian sus salidas.

El problema surge cuando la compuerta AND cuya salida va a cambiar a 0 realiza primero su transición. Un posible remedio en el que podría pensarse consiste en introducir un retardo adicional en esta trayectoria para retardar su transición, lo cual puede efectuarse, por ejemplo, con dos inversores consecutivos o un búfer. La lógica permanecería sin cambio; únicamente aumentaría el retardo. Sin embargo, la velocidad es un parámetro de diseño importante y se prefieren los circuitos con respuesta rápida en vez de los lentos.

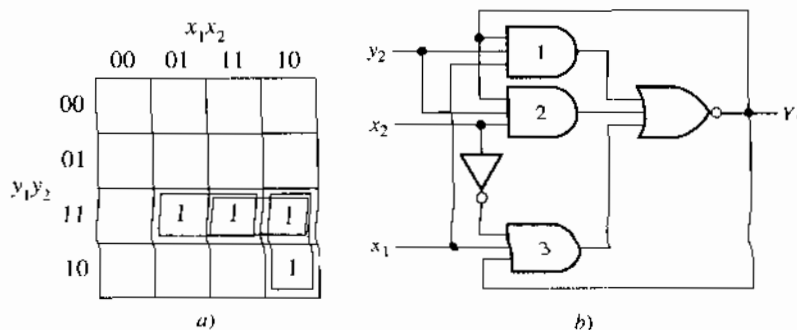


Figura 20. Adición de una compuerta para evitar un riesgo.

Considere que existe una tercera compuerta AND alimentando la compuerta OR de manera que su salida permanezca en 1 para ese cambio particular de la entrada primaria. La salida de la compuerta OR seguiría siendo 1, sin importar cuál de las otras compuertas efectuó primero su transición y tampoco qué tan largas fueron las transiciones. Lo anterior se consigue incluyendo otro implicante primo en la expresión, como se ilustra en la figura 20a. Ni la expresión ($Y_1 = x_1x_2'y_1 + x_1y_1y_2 + x_2y_1y_2$) ni el circuito son mínimos en este caso. Hay una compuerta redundante que no sirve a otro propósito más que el de evitar el riesgo.

Observamos aquí un ejemplo de dos requerimientos en conflicto: simplicidad del circuito y confiabilidad de operación. El circuito más simple es el mínimo, aunque éste es susceptible a los riesgos. Para eliminar estos últimos, incrementamos la complejidad del circuito. No debe suponerse a partir del ejemplo que *todos* los circuitos mínimos tendrán riesgos. Por ejemplo, observe el mapa para Y_{2nuevo} en la figura 17 y su implementación en la figura 18. Su expresión en (2) es la suma de tres implicantes primos. Cuando $y_1y_2 = 01$, el tercer implicante primo se mantiene igual a 1 para cualquiera cambios permisibles en las entradas primarias. En consecuencia, el circuito no tiene riesgos aun cuando sean mínimos.

Considerando sólo realizaciones de suma de productos de dos niveles, el principio general en el diseño de circuitos sin riesgos es asegurar que cualquiera minitérminos adyacentes para los cuales el valor de la función es 1 se cubran mediante un implicante primo. (Advierta que ésta es una condición suficiente sólo para que un circuito de dos niveles no tenga riesgo. La situación se torna más complicada para circuitos de orden superior. Es posible que éstos estén libres de riesgos sin que se satisfaga el principio anterior.)

EJEMPLO 3

El principio general se ilustra mediante la tabla de transición reducida de la figura 21a. Examinando la transición requerida a partir de cada estado estable para cambios de entrada permisibles, advertimos que se pide que cada estado sea adyacente a los otros tres estados. Esto es imposible de conseguir únicamente con dos variables de estados. En consecuencia, debe incrementarse el número de dichas variables. Una tabla de transición en la cual se dan a cada uno de los cuatro estados dos asignaciones diferentes se muestra en la figura 21b. Para esta asignación, cada cambio de entrada permisible lleva a una transición directa al estado estable requerido, sin ciclos. De este modo la transición se realiza en el menor tiempo de transición (TT) posible.⁹

⁹ También son posibles otras asignaciones en las cuales no se dan a todos los estados asignaciones múltiples. Eso deja algunos renglones con entradas de valor irrelevantes que es posible utilizar para formar ciclos. La implementación resultante tendrá un mayor tiempo de transición total.

					$Y_1 Y_2 Y_3$				
					$x_1 x_2$				$y_1 y_2 y_3$
					00	01	11	10	
S									
$x_1 x_2$									
	00	01	11	10					
1	①	2	①	3	1 → 000				
2	3	②	②	②	2 → 001				
3	③	4	2	③	3 → 011				
4	1	④	④	2	4 → 010				
					3 → 101				
					4 → 101				
					1 → 111				
					2 → 110				

a)

b)

a)

b)

Figura 21. Ejemplo de tabla de transición con asignaciones.

$y_1 = 0$					$y_1 = 1$			
$x_1 x_2$					$x_1 x_2$			
	00	01	11	10	10	11	01	00
00		1				1		
01	1	1	1	1	1	1	1	1
11	1		1	1	1		1	1
10								

Figura 22. Diseño sin riesgo de Y_3 en el ejemplo 3.

El mapa para Y_3 se muestra en la figura 22. (Confírmelo todo.) El orden de las variables para establecer los números de minterminos es $y_1 x_1 x_2 y_2 y_3$. Hay tres implicantes primos que tienen las siguientes listas de minterminos:

$$\begin{aligned}
 x_1 y_3 &= \Sigma(9, 11, 13, 15, 25, 27, 29, 31) \\
 x_2' y_3 &= \Sigma(1, 3, 9, 11, 17, 19, 25, 27) \\
 x_1' x_2 y_2' &= \Sigma(4, 5, 20, 21)
 \end{aligned} \quad (3)$$

La correspondiente expresión mínima s de p para la excitación es

$$Y_3 = x_1 y_3 + x_2' y_3 + x_1' x_2 y_2' \quad (4)$$

Nota: ciertos pares de minterminos son adyacentes, a saber, aquéllos para los cuales la excitación $Y_3 = 1$: 1 y 5, 17 y 21, 5 y 13 y 21 y 29. Un cambio de entrada permisible cambiará las secundarias de un miembro de cada par a las del otro. Así, *cada uno representa un riesgo*.

De acuerdo con el mapa, se ve que un conjunto de ocho minterminos forma un cubo de orden 3, aunque en la expresión para Y_3 , éstos no son cubiertos en su totalidad por un solo implicante primo. El remedio resulta claro: para eliminar los riesgos, debe agregarse a Y_3 un implicante primo que cubra a todos estos minterminos. La nueva expresión para Y_3 será:

$$Y_3 = x_1 y_3 + x_2' y_3 + x_1' x_2 y_2' + y_2' y_3 \quad (5)$$

(Dibuje el circuito representado por esta expresión.) A costa de una compuerta AND adicional, Y_3 se ha dejado sin riesgo, y no se ha sacrificado la velocidad en el proceso. ■

Después de esto se plantean varias preguntas. Suponga que se ha implementado una expresión de suma de productos sin riesgo. El circuito se dibuja observando aquellas celdas en un mapa lógico para el cual la función de excitación es 1 y permanecerá en esta misma condición durante un cambio de entrada permisible. ¿El circuito sin riesgo permanecerá del mismo modo con base en los 0? A pesar de que la demostración se encuentra más allá de nuestro objetivo, la literatura al respecto ha proporcionado la respuesta: sí. Esto significa que no debemos preocuparnos en cuanto a asegurar que los 0 de una función permanecerán iguales a 0 cuando se suponen así, siempre que ya hayamos asegurado que los 1 permanecerán siendo 1 cuando éstos se suponen así.

Lo inverso también es verdadero. Es posible encontrar una implementación de producto de sumas sin riesgo de una manera completamente dual asegurando que las celdas 0 en un mapa lógico que son adyacentes bajo cambios de entrada permisibles se cubren mediante un implicante primo. Esto asegura que cuando el valor de la función de excitación se desea mantener en 0 para un cambio de entrada permisible, no ocurrirán parpadeos momentáneos respecto a 1. Sin embargo, lo anterior también asegura que la función de excitación no tendrá una transición incorrecta de 1 a 0 cuando se supone que permanecerá en 1. Como es válido, en general, para circuitos combinatorios, uno de estos dos circuitos podría resultar más simple que el otro.

En este punto es necesaria una advertencia. Lo que no tiene riesgo es el *circuito* o la expresión específica de la cual el circuito es una realización. Si esta expresión se maneja de forma diferente (aunque equivalente) no hay razón para creer que también el resultado no implica riesgo.

Riesgos dinámicos

En la sección precedente se explicó el caso en la que una función lógica permanecerá invariable (en 1 o 0) para un cambio de entrada permisible. En otras ocasiones, se requiere que una función determinada *cambie* de valor (de 1 a 0 o de 0 a 1) respecto a un cambio de entrada permisible. En el mapa de la figura 22 se ilustra un ejemplo. Considere que el estado total es $y_1x_1x_2y_2y_3 = 00100$ y que x_1 cambia de 0 a 1. De acuerdo con el mapa, la excitación irá de 1 a 0. Sin embargo, es posible que la excitación realice la transición requerida de 1 a 0 pero que entonces presente una señal espuria (regrese a 1 antes de ponerse en el valor deseado de 0). Esto podría ocurrir debido a la retroalimentación, en virtud de las diferencias en el retardo a lo largo de diversas trayectorias en el circuito realizado. Si esto debe suceder, existe el peligro de que un segundo cambio (hacia el valor incorrecto de 1) pudiera volverse permanente. Aquí habría de nuevo un riesgo. Esta vez, debido a que ocurre para un *cambio* requerido en el valor de la función, recibe el nombre de *riesgo dinámico*.

Por fortuna, se ha demostrado en otros trabajos que no se necesitan medidas especiales para eliminar los riesgos dinámicos; si un circuito de dos niveles no presenta riesgos estáticos, no presentará tampoco riesgos dinámicos. En consecuencia, el remedio que se explicó para los riesgos estáticos será también cura para los dinámicos.

Riesgos esenciales

Tanto los riesgos estáticos como los dinámicos ocurren debido a la manera en que se implementa una tabla de transición determinada. Sin embargo, en algunas situaciones, es posible que exista un riesgo a partir de la propia naturaleza de las especificaciones de diseño, y no de las características del procedimiento de diseño del circuito. Esto es, la posibilidad de efectuar una transición incorrecta debido a diferencias en el retardo a lo largo de diferentes trayectorias quizás surja de la estructura de la tabla de flujo.

I_1	I_2		
(a)	b	I_1	I_2
c	(b)	(a)	b
(c)	d	c	(b)
-	(d)	(c)	(c)

a) b)

Figura 23. Posibilidades de riesgo esencial.

S					S				
x_1x_2					x_1x_2				
	00	01	11	10		00	01	11	10
a	(a), 0	b	-	g		(a), 0	b, 0	f, 0	(a), 0
b	c	(b), 0	h	-	ag → a	c, 0	(b), 0	f, 0	-
c	(c), 0	d	-	g	b	(c), 0	d, 0	-	f, 0
d	e	(d), 0	h	-	c	e, 1	(d), 0	f, 0	-
e	(e), 1	f	-	g	d	(e), 1	f, 0	-	f, 0
f	a	(f), 0	h	-	c	(e), 1	f, 0	-	f, 0
g	a	-	h	(g), 0	agh → f	a, 0	(f), 0	(f), 0	f, 0
h	-	f	(h), 0	g					

a) b)

Figura 24. Tablas de flujo que ilustran riesgos esenciales. a) Primitiva. b) Reducida.

Considere un estado estable en una tabla de flujo. Suponga que, en virtud de un solo cambio en una variable de entrada, el estado estable que se alcanza en una columna adyacente es diferente del estado estable que se consigue empezando desde el mismo estado inicial luego de tres cambios permisibles consecutivos de la misma variable de entrada. Se ha demostrado ya que, si hay al menos dos trayectorias de retroalimentación en el circuito, esta circunstancia puede provocar una transición errónea para ciertas combinaciones de retardos a lo largo de las trayectorias de retroalimentación. Debido a que este riesgo es una consecuencia de la naturaleza de la tabla de estado misma, recibe el nombre de *riesgo esencial*. En la figura 23 se presentan ejemplos de estructuras de tablas de flujo que dan lugar a riesgos esenciales. Los encabezados I_1 e I_2 se refieren a las entradas.

Los riesgos esenciales no son poco comunes. Ocurren en muchos circuitos prácticos y no es posible eliminarlos agregando compuertas, como sí puede suceder con los riesgos estáticos. La eliminación de un riesgo esencial requiere el ajuste cuidadoso de los retardos en lazo de retroalimentación específicos en un circuito. La facilidad para efectuar lo anterior se logra sólo con experiencia.

EJEMPLO 4

Un circuito en modo fundamental formará parte de una cerradura electrónica. Tiene dos entradas de nivel x_1x_2 y una salida de nivel z . El cerrojo se abre cuando $z = 1$. La "combinación" de entrada que abre la cerradura es

$$x_1x_2: 00 \ 10 \ 00 \ 10 \ 00$$

$$z: 0 \ 0 \ 0 \ 0 \ 1$$

El objetivo consiste en determinar la tabla de flujo primitiva, reducirla y luego observar si existe algún riesgo esencial.

La tabla de flujo primitiva se muestra en la figura 24a. Uno de los patrones de la figura 23 es visible aquí, de modo que existe un riesgo esencial. (Verifique esta afirmación.) Podría argumentarse que el patrón desaparecería si se redujera la tabla. En la figura 24b se presenta la tabla mínima reducida. Partiendo del estado estable \textcircled{a} para $x_1x_2 = 00$, si x_2 cambia hacia 1, el estado estable que se alcanza es \textcircled{b} ; pero si x_2 cambia de 0 a 1, luego vuelve a 0 y después regresa a 1 de nuevo, el estado estable que se alcanza es \textcircled{d} , es decir, un estado diferente. En consecuencia, existe un riesgo esencial. ■

Ejercicio 6. a. Confirme la tabla reducida de la figura 24b.

b. Encuentre todos los demás riesgos esenciales en esta tabla de flujo.

Respuesta¹⁰

RESUMEN Y REPASO DEL CAPÍTULO

Este capítulo abordó el tema de circuitos secuenciales que no tienen reloj para sincronizar las transiciones de estado, los cuales son asíncronos. (Los cerrojos y flip-flops básicos también son circuitos asíncronos.) Los estados se establecen mediante los inevitables retardos a través de las compuertas lógicas. Hay dos clases generales de cambios de estado en estos circuitos: aquellos que resultan de cambios de los niveles de entrada y los provocados por los pulsos de entrada. Los circuitos en los cuales ocurren los cambios de estado como consecuencia de cambios del nivel de entrada se dice que operan en el modo fundamental. Este capítulo trató sólo de este tipo de circuitos. Entre los temas se incluyeron:

- La tabla de flujo contra la tabla de estados en máquinas síncronas.
- Estados inestables y estados estables.
- Tablas de flujo primitivas.
- Asignación de salidas a estados inestables.
- Transiciones rápidas y lentas hacia un estado estable.
- Secuencias de entrada aplicables en circuitos en modo fundamental.
- Minimización de tablas de flujo en máquinas incompletamente especificadas.
- Pares de estados implicados.
- Compatibilidad e incompatibilidad de estados.
- La tabla de fusión.
- Determinación de conjuntos de estados compatibles.
- Determinación de compatibles máximos.
- Determinación de conjuntos cerrados de estados compatibles.
- Asignación de valores de variables secundarias y reducción de tablas de fusión.
- Tablas de transición de estado.
- Procedimientos de diseño para circuitos en modo fundamental.
- Carreras en circuitos en modo fundamental:
 - Críticas.
 - No críticas.

¹⁰ Existen otros cinco riesgos esenciales.

- Ciclos como transiciones entre dos estados estables.
- Oscilaciones que nunca conducen a un estado estable.
- Riesgos estáticos: cambios de valor momentáneos en valores secundarios.
- Riesgos estáticos: diseño sin riesgo.
- Riesgos dinámicos.
- Riesgos esenciales.

PROBLEMAS

Usted será capaz de efectuar partes de muchos de estos problemas sólo después de estudiar los temas finales del capítulo. Guarde las soluciones parciales; complete entonces los problemas después de que haya estudiado el material final.

1. I. Una máquina asíncrona en modo fundamental tiene dos entradas y una salida. La salida se convierte en 1 sólo después de la siguiente secuencia de entrada:

$$x_1x_2: 00 \rightarrow 10 \rightarrow 11 \rightarrow 01$$

Además, si x_1 y x_2 tiene el mismo valor, entonces x_2 no puede cambiar antes que x_1 .

- a. Construya una tabla de flujo primitiva.
- b. Suponiendo una salida rápida sin parpadeo, reduzca la tabla a una forma mínima.
- c. Efectúe una asignación válida y escriba expresiones para las funciones de excitación.
- d. Implemente el circuito.
- e. Analice si el circuito tiene riesgos esenciales.

II. Repita las partes anteriores con la modificación de que, después de que la salida se convierte en 1, sigue siendo 1 hasta que x_1 cambia de valor.

2. I. Una máquina asíncrona tiene dos entradas x_1 y x_2 y una sola salida z . La salida se convierte en 1 sólo cuando la combinación de entrada pasa de 01 a 11.

- a. Construya una tabla de flujo primitiva empezando desde el estado de borrado. Siga primero cualquiera secuencias de entrada que originen $z = 1$. Considere después cualquiera otros cambios de entrada permisibles a partir de los estados estables existentes y agregue nuevos estados como sea necesario para completar la tabla.
- b. Asigne salidas de estado inestables de manera que la máquina no tenga parpadeo.
- c. Obtenga una tabla mínima reducida. Si existe más de una posibilidad, elija aquella que tenga ventajas sobre las otras.
- d. Elija una asignación que no presente carreras críticas. (Si hay más de una posibilidad, intente cada una y señale la diferencia en la complejidad.)
- e. Escriba expresiones sin riesgo para las variables de estado y obtenga una realización de circuito.

II. Repita toda la parte I para el requerimiento de que la salida permanece invariable salvo en lo que respecta a los siguientes cambios de entrada:

$z = 1$ si la secuencia de entrada va de 10 a 11

$z = 0$ si la secuencia de entrada va de 11 a 10.

3. Una cerradura electrónica cuenta con dos entradas de nivel x_1 y x_2 y una sola salida de nivel z . La cerradura se "abre" cuando $z = 1$. La "combinación" que lo abre es la siguiente. Empezando en $x_1x_2 = 00$, x_1 se activa y desactiva (va hacia 1 y luego a 0) dos veces; después se activa x_2 . Cualquier cambio subsecuente en x_1 o x_2 cierra el reloj.

- a. Construya una tabla de flujo primitiva empezando desde el estado de borrado (00) y siguiendo una secuencia de entrada que abra la cerradura.
- b. Complete la tabla de flujo primitiva, incluyendo todos los demás estados posibles.
- c. Considere transiciones rápidas sin parpadeo y reduzca la tabla a la forma mínima.

- d. Realice una asignación válida sin carrera y escriba expresiones para las funciones de excitación.
 - e. Obtenga una realización para el circuito.
4. Una cerradura electrónica tiene tres entradas A , B y C y una sola salida z . La cerradura se abre sólo después de la siguiente secuencia empezando desde la condición de borrado (000):

A se activa y desactiva una vez; entonces B se activa; luego se activa C.

- a. Construya una tabla de flujo primitiva sin parpadeo. (Verifique si puede especificar el número de estados en la tabla antes de construirla; ¿estuvo usted en lo correcto?)
 - b. Reduzca la tabla de flujo a la forma mínima.
 - c. Especifique cualquier riesgos esenciales presentes en esta tabla.
 - d. Efectúe una asignación sin carrera crítica.
 - e. Construya una tabla de transición, especificando cualquier ciclo que se forme para evitar carreras críticas.
 - f. Obtenga expresiones sin riesgo para las variables de estado y salida.
 - g. Implemente el circuito correspondiente.
5. Un circuito secuencial en modo fundamental cuenta con dos entradas x_1, x_2 y dos salidas z_1, z_2 . Siempre que cualquier entrada cambia de 0 a 1 cuando la otra entrada es 0, una de las dos salidas cambia su valor; en otro caso, no hay cambio en la salida. Suponga que las salidas son al principio 00 y que la primera salida que cambiará es z_2 .
- a. Construya una tabla de flujo rápida sin parpadeo.
 - b. Reduzca ésta a una tabla mínima. Si hay más de una posibilidad, elija una con el número máximo de entradas sin especificar.
 - c. Seleccione una asignación válida y construya una tabla de transición.
 - d. Especifique cualquier ciclos que sea posible formar e indique el número de tiempos de transición implicados en cada uno.
 - e. Escriba expresiones sin riesgo para las variables de estado y expresiones para las funciones de salida.
 - f. Determine una realización de la expresión en la parte e.
6. Un circuito secuencial en modo fundamental tiene dos entradas x_1, x_2 y una sola salida z . La salida permanecerá en 0 hasta que aparece el último conjunto en la siguiente secuencia de entradas, cuando éste cambia a 1:

$$x_1x_2: 00 \rightarrow 10 \rightarrow 11 \rightarrow 01$$

- a. Construya una tabla de flujo sin parpadeo, siguiendo primero una secuencia de entrada correcta para producir $z = 1$.
 - b. Obtenga una tabla reducida mínima. Si hay más de una, elija aquella con el mayor número de entradas sin especificar.
 - c. Determine una asignación sin carrera crítica y expresiones sin riesgo para las variables de estado. Obtenga también una expresión para la salida.
 - d. Construya una implementación que realice las expresiones en la parte c.
 - e. En la parte b, suponga que los estados redundantes no se eliminan de los estados compatibles en la cobertura mínima cerrada. ¿Qué ventaja o desventaja puede existir en esto?
7. Un circuito asíncrono en modo fundamental tiene dos entradas x_1, x_2 y una sola salida z . La salida se convertirá en 1 si y sólo si la secuencia de entrada es cualquiera de las siguientes:

$$x_1x_2: 00 \rightarrow 10 \rightarrow 11 \quad \text{o} \quad 11 \rightarrow 01 \rightarrow 11$$

- a. Construya una tabla de flujo primitiva empezando desde el estado de borrado. Siga primero las secuencias de entrada que producen $z = 1$; continúe después con todos los demás cambios de entrada permisibles desde los estados estables ya establecidos y agregue nuevos estados, según sea necesario, para completar la tabla.
- b. Complete la tabla suponiendo salidas rápidas sin parpadeo.

- c. Reduzca a una tabla de flujo mínima.
- d. Elija una asignación válida y construya una tabla de transición. Verifique que no haya carreras críticas.
- e. Realice el circuito.

8. Una tabla de flujo parcialmente reducida se indica en la figura P8.

		S x_1x_2			
		00	01	11	10
a	(a), 0	b, 1	—	d, 0	
b	(b), 1	(b), 1	c, 1	e, 1	
c	—	b, 1	(c), 1	e, 1	
d	a, 0	b, —	—	(d), 0	
e	a, 0	(e), 0	f, 0	(c), 1	
f	—	e, 0	(f), 0	d, 0	

Figura P8

- a. Encuentre una tabla reducida mínima.
- b. Si es posible, elija una asignación válida y construya una tabla de transición. Si no es posible, incremente el número de variables de estado y elija una asignación válida que produzca asignaciones múltiples para algunos o todos los estados.
- c. En el último caso, el número de variables de estado será el mismo que el número necesario para la tabla original. Elija una asignación válida para la tabla original (antes de la reducción) y construya una tabla de transición. Si existen algunos ciclos, especifíquelos. Compare la complejidad de la realización.

		S x_1x_2			
		00	01	11	10
(a)	(a)	(a)	c		
a	c	(b)	(b)		
(c)	(c)	(c)	d		
(d)	a	b	(d)		
d	(e)	c	(e)		
a	(f)	(f)	b		
		a)			

		S x_1x_2			
		00	01	11	10
(a)	c	f	(a)		
d	(b)	c	a		
(c)	(c)	e	(c)		
(d)	f	(d)	c		
a	b	(e)	f		
c	(f)	(f)	(f)		
		b)			

		S x_1x_2			
		00	01	11	10
(a)	d	b	(a)		
(b)	c	(b)	c		
a	(c)	d	(c)		
b	(d)	(d)	a		
		c)			

		S x_1x_2			
		00	01	11	10
(1)	4	(1)	(1)		
(2)	(2)	(2)	1		
5	4	(3)	6		
2	(4)	3	5		
(5)	2	1	(5)		
1	(6)	2	(6)		
		d)			

Figura P9

9. Para cada tabla en la figura P9:

- Encuentre una asignación válida que no tenga carreras críticas y que requiera el menor número de variables secundarias.
- Construya una tabla de transición y escriba expresiones para las variables de estado. Encuentre un circuito implementado para cada caso.

	$\begin{array}{c} Y_1 Y_2 \\ x_1 x_2 \end{array}$			
$y_1 y_2$	00	01	11	10
a \rightarrow 00	00	01	00	00
b \rightarrow 01	00	01	11	01
c \rightarrow 11	10	11	11	10
d \rightarrow 10	00	10	10	00

Figura P10

10. Una tabla de transición sin carrera se muestra en la figura P10.

- Obtenga una implementación de circuito.
 - Suponga un estado total inicial de $x_1 x_2 y_1 y_2 = 0000$ seguido de un cambio de entrada hacia $x_1 x_2 = 01$. Tomando en cuenta los retardos de compuerta (supuestos iguales), evalúe todas las salidas de compuerta iniciales y los cambios en estas salidas. Verifique que se efectúa la transición de estado correcta.
 - Repita la parte b empezando en el estado total 11 (11) seguido por un cambio de entrada hacia $x_1 x_2 = 10$.
11. Encuentre una implementación sin riesgo de producto de sumas de la función de la figura 20 y compare su complejidad con la implementación sin riesgo de suma de productos en esa figura.
12. Encuentre una implementación mínima de producto de sumas de la función cuyo mapa se indica en la figura 22. ¿Esta no presenta riesgo? Compare su complejidad con la de la expresión de suma de productos sin riesgo en (5)

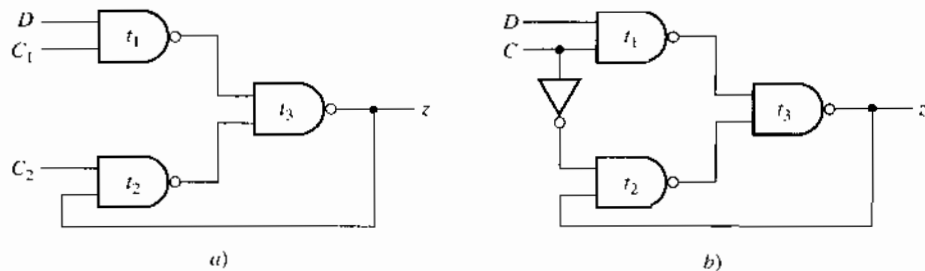


Figura P13

13. El circuito que se muestra en la figura P13a tiene el propósito de ser un cerrojo. D es la entrada de datos; C_1 y C_2 son las entradas de reloj; y t_1 , t_2 y t_3 son los retardos de propagación de las compuertas. Cuando $C_1 C_2 = 10$, el cerrojo es *transparente*, o abierto; cuando $C_1 C_2 = 01$, el cerrojo está *reteniendo*, o cerrado. El flanco de 0 a 1 de C_1 y el flanco de 1 a 0 de C_2 se referirán como los flancos de *apertura* de los relojes (éstos abren el cerrojo; los otros dos flancos de C_1 y C_2 se denominarán los flancos de *cierre* de los relojes). Se persigue que las dos entradas de reloj sean complementarias, aunque para la operación “adecuada” del cerrojo, se necesita una pequeña separación entre los dos flancos de apertura y también entre los dos de cierre. La “operación adecuada” equivale a lo siguiente:

El circuito debe cerrar los datos en forma confiable. Esto es, si cualquier valor x se establece sobre D bastante antes de que se cierre el cerrojo, el valor de la salida y después de que se ha cerrado el cerrojo debe ser D .

Y no debe generar una señal espuria si el valor almacenado en el presente en el cerrojo se recarga en este último.

Esto es, si $y = z$ (donde z puede ser 0 o 1) y z se establece en D lo bastante antes de que se abra el cerrojo, y debe seguir siendo z sin ninguna salida incorrecta momentánea cuando el cerrojo se abre.

- Determine la separación adecuada entre los flancos de apertura de C_1 y C_2 y entre sus flancos de cierre. En cada caso, indique cuál de los relojes debe liderar. Explique, con la ayuda de formas de onda, la operación inadecuada que podría ocurrir si no se mantiene la separación requerida.
 - Con base en su respuesta en la parte *a*, explique por qué no es aconsejable obtener C_1 invirtiendo C_2 , o viceversa.
 - Demuestre que el circuito combinatorio de la figura P13b, sin incluir la retroalimentación, tiene un riesgo estático.
 - Muestre cómo puede eliminarse este riesgo estático con la ayuda de lógica adicional. Con ésta, demuestre que el circuito secuencial formado por la retroalimentación que se presenta en la figura es un cerrojo que opera adecuadamente, de acuerdo con la definición de operación adecuada que se señaló antes.
- Diseñe un circuito asíncrono con dos entradas (x y y) y una salida (z). Si x y y concuerdan ($x = y$), entonces $z = x = y$. En otro caso z retiene el valor cuando x y y concordaron. (Este circuito recibe el nombre del elemento *C* de Miller.)
 - Un circuito asíncrono tiene N entradas (x_1, \dots, x_n) y una salida z . Las entradas representan las opiniones (sí/no) de N personas respecto a una pregunta controversial. Si dos tercios o más de las personas tienen la misma opinión, z debe reflejarlo. En otro caso z debe reflejar la última opinión sostenida por dos tercios o más de las personas. Determine una realización de este circuito.
 - Un circuito de *retardo de control* es un circuito asíncrono con dos entradas, C y x , y una sola salida, z . C es un reloj —un tren de pulsos periódicos— y x corresponde a una secuencia de pulsos aperiódicos que tienen el mismo ancho que los pulsos de reloj pero retrasados respecto del pulso de reloj por una pequeña fracción de un ancho de pulso. La salida z es un pulso del mismo ancho que se inicia mediante un pulso x pero que se encuentra retrasado exactamente un periodo de reloj a partir del pulso x . (En realidad, un pulso x podría ser la salida de otro circuito de retardo de control.) En la figura P16 se muestra un diagrama de temporización. Considere que el estado de borrado ocurre en cualquier tiempo posterior a la ocurrencia de un pulso de salida y antes del siguiente pulso de reloj.

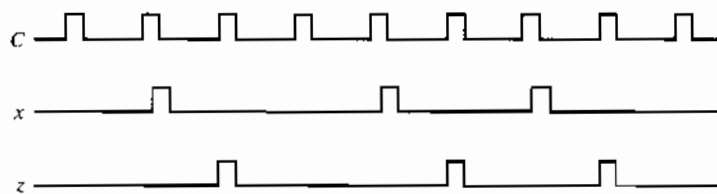


Figura P16

- Construya una tabla de flujo primitiva.
- Encuentre una tabla reducida mínima.
- Efectúe una asignación sin carreras críticas.
- Si hay carreras no críticas, se producirán diferentes tablas de transición cuando una o la otra secundaria cambie primero. Elija aquella que produzca las expresiones más simples para las variables de estado y realice el circuito.

	$Y_1 Y_2$ $x_1 x_2$			
	00	01	11	10
a \rightarrow 00	00	11	00	11
b \rightarrow 01	11	01	11	11
c \rightarrow 11	11	11	00	11
d \rightarrow 10	00	10	11	11

Figura P17

17. a. En la tabla de flujo de la figura P17 encuentre todas las carreras, si las hay; especifique aquellas que son críticas y las que son no críticas.
 b. Suponga que la entrada total es $y_1 y_2 x_1 x_2 = 1101$ y que ésta cambia en 1111. Describa los cambios resultantes en las excitaciones. ¿Esto constituye un ciclo?
 c. Si es posible, encuentre otra asignación que no contenga carreras críticas. (Describa el proceso mediante el cual se obtiene una nueva asignación de este tipo o se concluye que esto no es posible.) Si encuentra cualquiera carreras no críticas y ciclos en la nueva asignación, especifíquelas.
18. La tabla de estados de una máquina asíncrona incompletamente especificada se presenta en la figura P18.

PS	NS, z $x_1 x_2$			
	00	01	11	10
A	-	G, 1	C, 1	G, 1
B	F, 1	-	E, 0	D, 0
C	-	G, 1	E, 0	-
D	-	G, -	-	D, 0
E	B, 1	-	C, 1	-
F	B, 1	-	A, 0	A, 0
G	-	G, 1	-	A, 0

Figura P18

- a. Construya una tabla de fusión.
 b. Determine una tabla reducida mínima que cubra la tabla dada. Cuando exista la posibilidad de elección de estados para incluir una compatible, explique las ventajas y desventajas de la elección.
 c. Efectúe una asignación de estados óptima y construya una tabla de transición.
 d. Suponiendo una implementación con flip-flops D , obtenga expresiones para las excitaciones de flip-flop.
 e. Realice el circuito.
 f. Repita las partes d y e si los flip-flops son JK .

Estado, z			
x_1x_2			
00	01	11	10
$\textcircled{A}, 0$	B, 0	—	D, 0
A, 0	$\textcircled{B}, 0$	C, 0	—
—	B, 0	$\textcircled{C}, 0$	D, 0
A, 0	—	E, 1	$\textcircled{D}, 0$
—	B, 0	$\textcircled{E}, 1$	F, 1
G, 1	—	E, 1	$\textcircled{F}, 1$
$\textcircled{G}, 1$	H, 1	—	F, 1
G, 1	$\textcircled{H}, 1$	F, 1	—

Figura P19

19. La tabla de flujo primitiva para un circuito secuencial asíncrono se presenta en la figura P19.

- Construya una tabla de fusión.
- De todas las coberturas cerradas mínimas, elija la que con mayor probabilidad permita una asignación sin carrera y efectúe una asignación sin carrera; construya una tabla de flujo reducida.
- ¿Es posible efectuar una asignación que no tenga carrera sin aumentar el número de variables de estado? Si es así, efectúe tal asignación y construya una tabla de transición; especifique el número de ciclos que usted haya formado para evitar carreras críticas.
- Si no, incremente el número de variables de estado y repita la parte c.
- Implemente la tabla apropiada.

Capítulo 8

Diseño con lenguajes de descripción de hardware

Los capítulos precedentes presentaron los principios básicos del diseño lógico digital. (Álgebra de conmutación, mapas lógicos, diagramas de estado, tablas de transición, etc.) Aunque se describieron algunos procedimientos algorítmicos, y exista software para ejecutar estos algoritmos, no se utilizaron herramientas. Los procedimientos manuales que se expusieron en los capítulos anteriores resultan inadecuados para tratar con máquinas que necesitan más de, digamos, cuatro o cinco flip-flops (16-32 estados).

En máquinas más grandes, el número de asignaciones de estado que habría de considerar sería gigantesco; la derivación de los requerimientos de excitación de flip-flop requeriría mapas lógicos de más de seis variables. A pesar de eso, estos principios forman las bases indispensables en las herramientas de diseño asistido por computadora (CAD) utilizadas en el diseño de sistemas lógicos a gran escala.

Antes de la construcción de un prototipo, es una práctica estándar de los ingenieros crear una especificación legible para la computadora de un diseño que puede analizarse con programas de simulación. Este método permite la verificación de la corrección funcional de un diseño en mucho menor tiempo y esfuerzo en comparación con la evaluación de un prototipo. Hasta aproximadamente 15 años antes del fin de siglo, una elección popular para la especificación de un sistema digital fue el *ingreso de diseño mediante diagrama esquemático* (denominada a menudo captura de esquemático). El proceso del ingreso de diseño mediante diagrama esquemático supone que el diseñador ha efectuado ya la síntesis de diseño, debido a que el esquemático que se está capturando es en realidad una implementación.

Así, la captura de esquemático se limita al diseño de sistemas hasta con el orden de 100 compuertas y flip-flops (sistemas que pueden ser sintetizados en forma manual).

Los límites de la captura de esquemático se reconocieron de inmediato, y fue necesaria una alternativa para la especificación de sistemas digitales. Un requerimiento para esta alternativa es que permita al diseñador especificar el comportamiento sin efectuar la síntesis manual. Los *lenguajes de descripción de hardware (HDL)* evitan problemas con la captura de esquemático y la síntesis manual y en la actualidad se usan universalmente en el diseño.

El principal beneficio de especificar un circuito lógico digital utilizando un HDL es que el diseñador sólo necesita capturar la *especificación* del circuito en un lenguaje de este tipo, no el circuito mismo. En consecuencia, es posible llevar a cabo la implementación de una manera automatizada, mediante el uso de herramientas CAD.

Un lenguaje de descripción de hardware consiste en una secuencia de sentencias en un lenguaje de comandos, muy similar a un programa de software en un lenguaje de alto nivel. Sin em-

bargo, el hardware opera de manera inherente en paralelo, de manera tal que las herramientas que usan el lenguaje, como simuladores, sintetizadores y traductores, deben interpretarlo en forma apropiada. Además, para escribir especificaciones correctas, el diseñador debe entender cómo interpretan el lenguaje estas herramientas.

El uso de un HDL tiene otro beneficio. Podemos concebir dos métodos en el diseño:

- Efectuar directamente una implementación del hardware a partir de las especificaciones, o
- Si es posible, simular el diseño para verificar su corrección antes de que se produzca el hardware.

Evidentemente, es preferible lo último. Cuando una especificación del sistema digital se escribe en un HDL, es posible ejecutarla utilizando un simulador para verificar el funcionamiento correcto y los requerimientos de temporización. Sólo en ese caso se implementa el hardware. De esta manera se consiguen enormes ahorros en esfuerzo y costo.

Asimismo, una vez que se verifica la especificación HDL respecto al funcionamiento correcto, se cuenta con herramientas de síntesis para efectuar un diseño a nivel de compuertas. Este tipo de herramienta de síntesis aplica métodos similares a los que se estudiaron en los capítulos 3 y 6.¹ Por último, la especificación a nivel de compuertas se traduce en un código que puede utilizarse para programar un dispositivo integrado específico.

En la actualidad hay muchos lenguajes de descripción del hardware en uso, algunos más populares que otros. Dos de los más populares en el uso profesional son VHDL² y Verilog.³ Cada uno de estos lenguajes es completo. Sin embargo, presentar uno de estos lenguajes requiere plasmarlo en un libro —para que usted asimile los detalles de manera suficiente y los use en el diseño—, lo cual no es recomendable en este nivel.

1 EL LENGUAJE DE DESCRIPCIÓN DEL HARDWARE ABEL

En este libro utilizaremos el lenguaje de descripción del hardware que tiene el acrónimo ABEL⁴ (Lenguaje de expresión booleana avanzado). Aunque no es tan completo como VHDL o Verilog, ABEL es conceptualmente similar y se usa de manera amplia en la especificación de sistemas realizados con dispositivos lógicos programables (PLD).

Presentaremos en forma gradual el lenguaje ABEL, con los detalles suficientes para permitirle usarlo en el diseño de circuitos recurriendo a PLD.⁵ Luego de que usted haya iniciado la práctica como ingeniero, su familiaridad con ABEL le permitirá aprender y aplicar con facilidad alguno de los otros lenguajes de descripción del hardware.

ABEL utiliza palabras reservadas tales como pin (*patillas*) y node (*nodos*) para representar puntos en un circuito donde se realizan conexiones; emplea ecuaciones, tablas de verdad y diagramas de estados para la especificación del comportamiento del circuito. Comenzaremos no con la descripción de los detalles del lenguaje, sino con ejemplos, de los cuales, al analizar sus respectivas especificaciones, nos permitirá identificar características generales del lenguaje.

¹ Los métodos utilizados en las herramientas de síntesis contemporáneas son más avanzados que los que estudiamos en este libro, aunque son extensiones de la misma teoría.

² VHDL fue creado a través del programa patrocinado del VHSIC (siglas en inglés de circuito integrado de muy alta velocidad) del Departamento de Defensa de los Estados Unidos; es un lenguaje público. VHDL es una jerarquía de acrónimos: quiere decir Lenguaje de Descripción de Hardware VHSIC.

³ Verilog es un lenguaje de descripción de hardware creado por Cadence, Inc., a principios de la década de los años ochenta y ha sido desde entonces un lenguaje público.

⁴ Data I/O Corporation creó ABEL en 1984.

⁵ En cuanto a características y detalles de lenguajes adicionales, véase David Pellerin y Michael Holley, *Digital Design Using ABEL*. Prentice Hall, 1994.

```

(1) module sumador
(2) Title 'celda de sumador completo'

(3) Declarations

(4) A PIN;
    B PIN;
    Cin PIN;
(5) S PIN istype 'com'; "salida combinatoria
    Cout PIN istype 'com';

(6) Equations

(7) S = A $ B $ Cin; "salida suma
    Cout = A & B # A & Cin # B & Cin; "salida del acarreo

(8) end sumador

```

Figura 1. Descripción ABEL de un sumador completo.

Especificación del sumador en ABEL

En la figura 1 se muestra la descripción ABEL de un sumador completo. (Repase el capítulo 4 acerca de sumadores si lo requiere.) Se asemeja a un programa escrito en un lenguaje de programación de alto nivel, aunque es diferente en el aspecto conceptual. Los números de línea que se muestran entre paréntesis no constituyen una característica de lenguaje, sino que se indican, sólo en este ejemplo, para una fácil referencia. (Esta numeración se omitirá en descripciones ABEL subsecuentes.)

El término pin se usa con dos interpretaciones. Una es la ubicación física en un dispositivo al cual se hace una conexión externa; la otra es una señal que se asocia con esa ubicación. Es análogo a una variable en un lenguaje de programación. Las ecuaciones especifican el comportamiento del circuito, donde la señal de salida está en el lado izquierdo y la expresión que produce la señal de salida en el derecho. Estas ecuaciones son análogas a las sentencias de asignación en un lenguaje de programación.

La principal diferencia entre un HDL y un lenguaje de programación es que las sentencias en este último se valúan en secuencia. Por otro lado, las sentencias en el lenguaje de descripción de hardware se interpretan como si la ejecución se efectuara paralelamente. El circuito lógico combinatorio para un sumador completo, por ejemplo, no produce la suma primero y luego el acarreo, sino que los genera simultáneamente. El hardware opera de manera inherente en paralelo; en consecuencia, un HDL debe caracterizar este paralelismo. En la descripción ABEL de la figura 1 no se distingue cuál ecuación (S o Cout) se escribe primero, pues la evaluación mediante un simulador es como si éstas se ejecutaran en paralelo.

Las entradas en una descripción ABEL de una tarea lógica reciben el nombre de *sentencias*. Una necesidad obvia corresponde a las sentencias de inicio y término. El inicio es una *expresión module*, en el cual se le da un nombre al módulo. La línea (1) muestra este identificador como module sumador, sin comillas ni nada que lo resalte. El enunciado de término tiene la forma *end nombre*; para el sumador, la línea (8) muestra esto como end sumador, también sin comillas o letras mayúsculas. También es necesario una sentencia de título (por razones que se explicarán adelante); la línea 2 muestra lo anterior como Title 'celda de sumador completo'. Puede haber cualquier número de *sentencias de comentarios*, los cuales ignoran los compiladores del lenguaje.

Símbolo del Operador	Descripción del Operador
!	NOT
&	AND
#	OR
\$	XOR
!\$	XNOR

Figura 2. Algunas operaciones lógicas en ABEL.

Éstos empiezan y terminan con una doble comilla o puedan terminar también con el final de la línea. Su propósito principal es esclarecer todo lo que indica la descripción ABEL.

Las secciones principales de una descripción ABEL son la sección de *declaraciones*, que se inicia con la palabra *Declarations*, como en la línea (3), y la sección de *ecuaciones*, que principia con la palabra *Equations*, como en la línea (6), ambas sin puntuación. La sección de declaraciones contiene las especificaciones de

- Las patillas de entrada y salida del circuito;
- Definiciones de nodos internos.

En este ejemplo no hay nodos internos.

En la figura 1, la descripción especifica las patillas de entrada como A, B y Cin y especifican las patillas de salida como S y Cout.

Para especificar un atributo de la señal, se usa la terminología *istype*. Así, el enunciado *istype 'com'* especifica las salidas como combinatorias. Las sentencias de declaración se terminan con un punto y coma. (Confirme todo lo anterior con referencia en la figura 1.)

Advierta las palabras que siguen al punto y coma en la línea 5: ¿cuál es su propósito? La línea de ecuaciones utiliza algunos símbolos que se describirán en lo que sigue.

Ejercicio 1. Revise de nuevo la figura 1. A partir de esta inspección, ¿puede usted imaginarse por qué se necesita tanto un nombre (dado en la expresión *module*) como un título? ¿Qué papel desempeña cada uno? ♦

Los símbolos que se utilizan en ABEL para los operadores lógicos son diferentes de los usuales. Los que se obtienen en ABEL se resumen en la tabla de la figura 2. ¿Cuántos de éstos se utilizan en la figura 1?

Ejercicio 2. Escriba sentencias de asignación ABEL para las funciones de propagación y generación de un sumador de acarreo anticipado. Reescriba las sentencias de asignación para la suma y el acarreo en la figura 1 utilizando las funciones de propagación y generación. ¿Este nuevo conjunto de sentencias describe el mismo comportamiento que el descrito en la figura 1? ♦

Descripción de comportamiento contra descripción operacional

Observemos de nuevo la descripción ABEL en la figura 1. Los detalles aclaran que el diseñador ya ha sintetizado la representación lógica del circuito. Podríamos denominar a ésta una descripción *operacional*.

En contraste, una descripción de *comportamiento* describe simplemente el comportamiento del circuito, no las operaciones que se ejecutan. El poder de un HDL resulta evidente cuando efectúa un diseño a partir de una descripción de comportamiento de este tipo.

Ilustraremos la diferencia entre una descripción operacional y una de comportamiento por medio de descripciones ABEL de un multiplexor 2 a 1. Este dispositivo tendrá dos entradas de

<pre> module 2to1mux Title 'multiplexor 2 a 1' Declarations D0 PIN; D1 PIN; S PIN; Y PIN istype 'com'; Equations Y = D0 & !S # D1 & S; end 2to1mux a) </pre>	<pre> module 2to1mux Title 'multiplexor 2 a 1' Declarations D0 PIN; D1 PIN; S PIN; Y PIN istype 'com'; Equations When S == 0 then Y = D0; Else Y = D1; end 2to1mux b) </pre>
---	---

Figura 3. Diferentes descripciones ABEL de un multiplexor 2 a 1.

datos, una entrada seleccionada y una salida de datos. En la figura 3 se muestran dos descripciones ABEL de tal circuito.

La descripción en la figura 3a requiere que el diseñador especifique, en la sección de ecuaciones, las operaciones lógicas detalladas de una implementación particular de un multiplexor 2 a 1. La descripción en la figura 3b, por otro lado, requiere sólo que el diseñador entienda el comportamiento (o especificación funcional) del multiplexor. Esto implica que uno no necesita conocer de manera exacta cómo implementar un sistema para crear una especificación inicial y formal cuya corrección se pueda verificar después.

Hay una gran ventaja al verificar la corrección de una especificación *de comportamiento* de un sistema respecto a hacerlo con una especificación *operacional detallada*. Puesto que la especificación se puede escribir sin detalles de la implementación, la primera requiere por lo general menos tiempo. Además, una vez que se ha verificado la especificación de comportamiento respecto a la corrección, pueden explorarse diversas implementaciones diferentes para optimizar características del sistema como la velocidad, la potencia, el tamaño o el costo.

Una especificación de comportamiento establece una referencia a partir de la cual es posible proceder con la síntesis de diseño.

Existen dos requerimientos para este tipo de especificación: debe ser formal, de manera que no haya ambigüedad en su interpretación, así como funcionalmente completa. Una descripción de lenguaje natural no puede interpretarse de manera automática, por lo que resulta necesario un lenguaje formal. Ésta es la motivación para los lenguajes de descripción de hardware.

En ABEL existen tres mecanismos para especificar el comportamiento del sistema: ecuaciones, tablas de verdad y diagramas de estados. Las ecuaciones y las tablas de verdad se usan ya sea en lógica combinatoria o secuencial; las descripciones de diagramas de estados se aplican evidentemente sólo en la última.

Las descripciones ABEL que se presentan en las figuras 1 y 3a utilizan ecuaciones para describir los circuitos. El símbolo = se usa siempre que se hace la asignación a una señal de salida combinatoria.

Como ya se mencionó, también es posible describir un circuito por medio de su tabla de verdad. La figura 4 ilustra lo anterior para el sumador completo y el multiplexor 2 a 1.

Puesto que la tabla de verdad puede traducirse en diversas realizaciones, una descripción ABEL de tabla de verdad corresponde a una descripción de comportamiento. Las tablas de verdad, sin embargo, son largas; a partir de ellas, se vuelve difícil reconocer la función que efectuará una descripción. Por ejemplo, la descripción del circuito que se da en la figura 3b se reconoce

```

module sumador
Title 'celda de sumador completo'

```

Declarations

```

A PIN;
B PIN;
Cin PIN;
S PIN istype 'com';
Cout PIN istype 'com';

```

Equations

```

TRUTH_TABLE ([A, B, Cin] -> [S, Cout])
[0,0,0] -> [0,0];
[0,0,1] -> [1,0];
[0,1,0] -> [1,0];
[0,1,1] -> [0,1];
[1,0,0] -> [1,0];
[1,0,1] -> [0,1];
[1,1,0] -> [0,1];
[1,1,1] -> [1,0];

```

```

end adder

```

a)

```

module mux2a1
Title 'multiplexor 2 a 1'

```

Declarations

```

D0 PIN;
D1 PIN;
S PIN;
Y PIN istype 'com';

```

Equations

```

TRUTH_TABLE ([S, D1, D0] -> [Y])
[0,0,0] -> [0];
[0,0,1] -> [1];
[0,1,0] -> [0];
[0,1,1] -> [1];
[1,0,0] -> [0];
[1,0,1] -> [0];
[1,1,0] -> [1];
[1,1,1] -> [1];

```

```

end mux 2 a 1

```

b)

Figura 4. Descripción ABEL utilizando la notación de tabla de verdad. a) Sumador. b) Multiplexor.

fácilmente como un multiplexor, pero no sucede lo mismo con la función de la descripción de la figura 4b. Por tanto, las descripciones de tabla de verdad resultan adecuadas sólo en diseños de tamaño modesto.

Especificación del sumador en ABEL

La descripción ABEL de una celda de sumador de un solo bit se mostró en la figura 1. Para un sumador de múltiples bits hay numerosas maneras de escribir la descripción; la forma más abstracta consiste en utilizar el operador de adición disponible en el lenguaje ABEL. La descripción de un sumador de 4 bits se muestra en la figura 5. No hemos introducido aún parte de la notación aquí; ahora corregiremos esa situación. Cuando un grupo de señales forma un canal o bus, se utiliza la notación A3..A0 para designar el grupo.⁶

Cualquier subsecuencia (por ejemplo, A2A1) de la secuencia A3..A0 puede ser referida como un grupo, o incluso como señales individuales (por ejemplo, A2). La notación [Cout,S3..S0] se usa para agrupar señales en un bus; en este caso la señal Cout es el bit más significativo del bus, aunque no tiene que ser así. La sintaxis .x. en [.x.,A3..A0] se usa para "llenar espacio" de manera que todos los argumentos de la ecuación sean del mismo tamaño. Es posible interpretar la sintaxis como si no hubiera conexión en la señal de bus correspondiente para este argumento.

⁶ En lenguaje ordinario, un autobús (bus en inglés) es un vehículo que transporta muchos pasajeros. Por analogía, en sistemas digitales, más que indicar una línea independiente para cada señal, se usa una línea gruesa en los diagramas para representar un *bus*, un vehículo hipotético para transportar una colección de señales. El término *bus* también se usa para referir la propia colección de señales.

```

module sumador_be
Title 'sumador_be'
"Descripción de comportamiento de sumador de 4 bits.

Declarations
A3...A0 PIN;
B3...B0 PIN;
S3...S0 PIN istype 'com';
Cout PIN istype 'com';
Cin PIN;
@ carry 2
Equations

when Cin == 1 then
    [Cout, S3..S0] = [.x., A3..A0] + [.x., B3..B0] + 1;
else
    [Cout, S3..S0] = [.x., A3..A0] + [.x., B3..B0];

end sumador_be

```

Figura 5. Descripción ABEL de un sumador de 4 bits.

Símbolo del operador	Descripción del operador	Tipo de operador
==	Comparación de igualdad	Relacional
!=	Comparación de desigualdad	
<	Menor que	
>	Mayor que	
<=	Menor o igual que	
>=	Mayor o igual que	Aritmético
-	Negación (complemento a 2)	
-	Resta	
+	Suma	
*	Multiplicación	
/	División	
%	Módulo	
<<	Corrimiento a la izquierda	
>>	Corrimiento a la derecha	

Figura 6. Operadores ABEL adicionales.

La ecuación

$$[Cout, S3..S0] = [.x., A3..A0] + [.x., B3..B0]$$

especifica una suma de 4 bits que puede producir un resultado de 5 bits.

Una *herramienta de síntesis* de ABEL traduce una descripción ABEL en una que es adecuada para la implementación. Por omisión, la herramienta de síntesis genera una implementación paralela máxima de dos niveles.

Así, la implementación que se genera para la descripción de un sumador corresponde a un sumador de acarreo anticipado.

Es posible generar una implementación de acarreo en cascada del sumador sin especificar de manera explícita las ecuaciones lógicas. Lo anterior se efectúa en la parte de declaraciones de la descripción dando la directiva *@carry*. Esta directiva tiene un argumento numérico que especifica el número de etapas de bit entre las propagaciones del acarreo. De tal manera, *@carry 1* especifica un sumador de acarreo en cascada, y *@carry 2*, un sumador donde los acarreos de grupos de 2 bits se calculan en paralelo y se conectan en cascada entre los grupos. La instrucción *@ carry* también es útil en otros circuitos aparte de los sumadores. En general, le indica a la

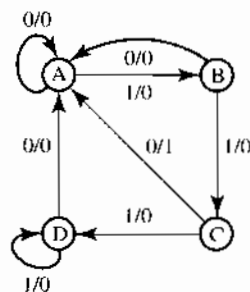


Figura 7. Diagrama de estados para el detector de secuencia 0110.

herramienta de síntesis que genere un circuito multinivel. Se pueden obtener numerosas directivas en ABEL.

Sólo se han usado hasta ahora unos cuantos de los operadores que se consiguen en ABEL. En la figura 6 se presentan operadores adicionales.

Ejercicio 3. Escriba la descripción de comportamiento de un comparador de magnitud con dos entradas de 4 bits y tres salidas de un solo bit. Una salida debe estar en el estado alto si las entradas son iguales, la otra estará en alto si el primer operando es mayor que el segundo, y la tercera salida estará en alto si es cierto lo opuesto. ♦

Especificación de circuito secuencial en ABEL

El siguiente paso en el estudio de ABEL implica prestar atención a los circuitos secuenciales. En lo que respecta a diagramas de estados, tablas de estados, condiciones de irrelevantes y aspectos similares, tal vez usted podría esperar mayor dificultad para producir una descripción ABEL para circuitos secuenciales. A medida que avancemos en el proceso, usted juzgará si éste es el caso.

Empezaremos considerando el detector de secuencias del ejemplo 1 en el capítulo 6. Dicho detector tiene una entrada y una salida; produce una salida de 1 siempre que se detecta la secuencia de entrada 0110. El diagrama de estado para este detector de secuencias de acuerdo con el capítulo 6, se muestra en la figura 7.

En la figura 8 se ilustra una descripción ABEL de esta máquina de estados finitos. El corazón de la descripción es la especificación del diagrama de estados junto con las transiciones de estado. Si es necesario (en el caso de entradas múltiples), la transición de estado puede ser una secuencia de enunciados *if, else if, else if..., else*.

Por ejemplo, si hubiese dos entradas para una máquina de estados, se escribiría:

```

If [x,y] == [0,0] then A
Else if [x,y] == [0,1] then B
Else if [x,y] == [1,0] then C
Else D
  
```

Las variables de estado Q1..Q0 deben declararse como istype 'reg' de manera que la herramienta de síntesis asignará flip-flops para su implementación.

Se utilizan dos términos para designar una señal. Una es NODE, la cual es una señal interna y no una salida del circuito. La otra es PIN que, como ya se explicó es una señal que se asignará a las patillas reales de entrada/salida de un dispositivo.

No hay mecanismo en ABEL para declarar nombres de estados simbólicos; entonces, las asignaciones de estados deben especificarse desde el principio. Una alternativa es no recurrir en absoluto a los nombres de estados simbólicos sino utilizar directamente las asignaciones de estado numéricas. La declaración [Q1..Q0].clk = clock especifica que clk es la señal que se conecta a la terminal de reloj de los flip-flops. La expresión .clk que acaba de señalarse constituye un

```

module detector
Title 'detector'
"detector de secuencia 0110

Declarations
clock, x PIN;
z PIN istype 'com';
Q1 .. Q0 NODE istype 'reg'; "salida registrada
A = [0, 0]; "asignación de estados
B = [0, 1];
C = [1, 0];
D = [1, 1];

Equations
[Q1 .. Q0] . clk = "definición de señal de reloj para salidas registradas
z=O1&!O0&!x;

state_diagram [Q1, Q0]
state A: if x then B else A; "transiciones de estado
state B: if x then C else A;
state C: if x then D else A;
state D: if x then D else A;

end detector

```

Figura 8. Descripción ABEL del detector de secuencia 0110 basado en el diagrama de estados.

sufijo de atributo correspondiente al nombre de la señal. Muchos más sufijos de atributo se usan además de .clk.

ABEL no tiene una forma explícita de distinguir señales que son entradas de aquellas que son salidas. (Es posible observar lo anterior en la figura 8 mediante las declaraciones clock y x; éstas no especifican si son o no entradas, y un compilador ABEL no puede determinar si corresponden a lo uno o a lo otro.) Para identificar una señal de manera explícita como una salida, se usan los atributos de señal 'com' y 'reg'.

Es posible recurrir a tablas de verdad o a ecuaciones para especificar condiciones de irrelevancia en una descripción ABEL. La especificación de valores irrelevantes utilizando tablas de verdad resulta simple. Cualesquiera combinaciones de entrada que se omiten en una tabla de verdad se supone que serán condiciones irrelevantes.

Además de la descripción de un detector de secuencia basado en un diagrama de estados como en la figura 8, puede escribirse una descripción utilizando la construcción de la tabla de verdad o las ecuaciones. En un circuito secuencial, la tabla de verdad corresponde a su tabla de transición. (Repase el capítulo 5 si lo considera necesario.) La descripción ABEL del detector de secuencias utilizando la construcción de la tabla de verdad se muestra en la figura 9.

Ejercicio 4. Suponga que el estado presente es B y que se recibe la secuencia de entrada 01001110110. Verifique que las descripciones en las figuras 8 y 9 producen la misma secuencia de salida. ♦

Advierta que la sintaxis para una tabla de verdad secuencial utiliza: > en lugar ->. La tabla de verdad en la figura 9 combina las tablas de transición y salida en una tabla de verdad. El sufijo de atributo .fb instruye a la herramienta de síntesis para que *retroalimente* la salida del registro Qstate hacia la entrada del decodificador del siguiente estado. En esta descripción,

```

module detector
Title 'detector'
"detector de secuencia 0110

Declarations
clock, x PIN;
z PIN istype 'com';
Q1 .. Q0 NODE istype 'reg'; "salida registrada
Qstate = [Q1, Q0];
A = [0, 0]; "asignación de estados
B = [0, 1];
C = [1, 0];
D = [1, 1];

Equations
[Q1 .. Q0] . clk = "definición de señal de reloj para salidas registradas
z = Q1 & !Q0 & !x;

truth_table ([x, Qstate.fb] => [Qstate -> [z])
[0, A] => [A] -> [0];
[1, A] => [B] -> [0];
[0, B] => [A] -> [0];
[1, B] => [C] -> [0];
[0, C] => [A] -> [1];
[1, C] => [D] -> [0];
[0, D] => [A] -> [0];
[1, D] => [D] -> [0];

```

Figura 9. Descripción ABEL del detector de secuencia 0110 utilizando una tabla de verdad.

Qstate.fb se interpreta como el estado presente y Qstate es el estado siguiente. Las expresiones de excitación del flip-flop se sintetizan de manera automática mediante la herramienta de mapeo a tecnología (que se describe más adelante) cuando se elige un dispositivo programable específico para la implementación.

Como ya se mencionó, la descripción del detector de secuencia 0110 también puede escribirse especificando las ecuaciones del siguiente estado. La descripción ABEL correspondiente se muestra en la figura 10. La desventaja de escribir la descripción con base en las ecuaciones

```

module detector
Title 'detector'
"detector de secuencia 0110

Declarations
clock, x PIN;
z PIN istype 'com';
Q1 .. Q0 NODE istype 'reg'; "salida registrada

Equations
[Q1 .. Q0] . clk = "definición de señal de reloj para salidas registradas
z = Q1 & !Q0 & !x;
Q1 := Q0.fb & x # Q1.fb & x;
Q0 := !Q0.fb & x # Q1.fb & x;

end detector

```

Figura 10. Descripción ABEL del detector de secuencia 0110 utilizando ecuaciones del estado siguiente.

```

module BCDto7seg
Title 'convertidor de código BCD a 7 segmentos'
Declarations
[D3 .. D0] PIN;
a PIN istype 'com';
b PIN istype 'com';
c PIN istype 'com';
d PIN istype 'com';
e PIN istype 'com';
f PIN istype 'com';
g PIN istype 'com';
Equations
a=D3&!D2&!D1#!D2&!D1&!D0#!D3&D2&D0#!D3&D1;
a?=D3&D2#D3&D1; "expresión que cubre las entradas irrelevantes
b=!D3&!D2#!D2&!D1#!D3&!D1&!D0#!D3&D1&D0;
b?=D3&D2#D3&D1;
c=!D3&D2#!D3&D0#!D2&!D1;
c?=D3&D2#D3&D1;
d=D3&!D2&!D1#!D2&!D1&!D0#!D3&D2&!D1&D0#!D3&D2&D1#!D3&D1&!D0;
d?=D3&D2#D3&D1;
e=!D3&D1&!D0#!D2&!D1&!D0;
e?=D3&D2#D3&D1;
f=D3&!D2&!D1#!D3&!D1&!D0#!D3&D2&!D1#D3&D2&!D0;
f?=D3&D2#D3&D1;
g=D3&!D2&!D1#!D3&D2&!D1#!D3&!D2&D1#!D3&D1&!D0;
g?=D3&D2#D3&D1;
end BCDto7seg

```

Figura 11. Descripción ABEL de un codificador BCD a siete segmentos utilizando ecuaciones para valores irrelevantes.

del siguiente estado es que la síntesis debe efectuarse primero para obtener las ecuaciones. Como se señaló antes, la ventaja más significativa de diseñar con un lenguaje de descripción de hardware es que es posible escribir una especificación formal y no ambigua y que la síntesis se puede efectuar posteriormente utilizando herramientas de software.

Condiciones irrelevantes en ABEL

Como ya se ha visto en los capítulos anteriores, las condiciones irrelevantes se pueden utilizar para minimizar el tamaño de la implementación de un sistema. Es posible que ABEL aproveche las condiciones irrelevantes, siempre que sea viable determinar a partir de la especificación que existen dichas condiciones.

Una especificación que usa ecuaciones como las que se indican en la figura 1 constituye una especificación completa. Las ecuaciones caracterizan combinaciones de entrada para las cuales la salida es 1 lógico; aquéllas para las cuales la salida es 0 lógico se encuentran implicadas.

¿Cómo especificamos condiciones irrelevantes cuando se describe un sistema utilizando ecuaciones? Por ejemplo, en la descripción ABEL de un decodificador BCD a siete segmentos utilizando una especificación de tabla de verdad, se omitirían los renglones de esta tabla correspondientes a los números de 4 bits del 10 al 15. Por otra parte, para especificar valores irrelevantes utilizando ecuaciones, uno debe escribir ecuaciones separadas empleando el operador asignación $?=$ o $?:=$. Estas ecuaciones representan coberturas de las combinaciones de entrada que corresponden a valores irrelevantes. La especificación de un decodificador BCD a siete segmentos utilizando ecuaciones se ilustra en la figura 11.


```

module 2to1mux
interface (D1, D0, S->Y);
Title 'multiplexor 2 a 1'
Declarations

D0 PIN;
D1 PIN;
S PIN;
Y PIN istype 'com';

Equations
When S == 0 then
    Y = D0;
Else
    Y = D1;
end 2to1mux

```

Figura 12. Declaración de interfaz en la descripción ABEL de la figura 3b que permite la instanciación en otros módulos ABEL.

Es necesario tener cuidado al especificar una ecuación expresando una condición irrelevante de modo que esto no incluya la ecuación correspondiente que especifica una salida de 1 lógico. En otras palabras, una ecuación que describe los casos de 1 lógico para la señal *a* no debe cubrir alguna de los valores irrelevantes de las entradas. En este sistema particular, advierta que las ecuaciones que cubren los valores irrelevantes de las entradas son las mismas para todas las señales de salida.

Especificaciones jerárquicas en ABEL

Hasta ahora, los módulos ABEL considerados han sido autocontenidos; ningún módulo ha dependido de otro. Al crear un módulo ABEL, sin embargo, es posible utilizar otros módulos ABEL en su descripción.

Cuando el módulo ABEL 1 se usa en el módulo ABEL 2, se dice entonces que 1 será instanciado en 2. Es posible afirmar también que el módulo 2 contiene una instancia del módulo 1. No hay un límite en el número de veces que un módulo puede utilizarse (instanciado). Esto permite una descripción jerárquica en la cual un bloque ABEL para una función se utiliza dentro de otro bloque ABEL. Esta característica de lenguaje promueve el uso de diseños de unidades pequeñas como bloques dentro de bloques mayores. La complejidad de describir sistemas grandes se reduce de esa manera.

Un módulo ABEL que se use dentro de otro módulo ABEL requiere una sentencia interfaz, como se muestra en la segunda línea de la descripción del multiplexor en la figura 12. La declaración de interfaz especifica todas las entradas y salidas a las que se tendrá acceso desde otro módulo.

Este multiplexor 2 a 1 puede utilizarse para construir un multiplexor 4 a 1, como se indica en la figura 13. El multiplexor 4 a 1 emplea tres multiplexores 2 a 1 conectados como se muestra en el diagrama esquemático de la figura 14. El enunciado de bloque funcional en la figura 13 especifica el módulo ABEL que se utilizará y los nombres de las instancias. Cada instancia del módulo utilizado requiere tener un nombre distinto de manera que sea posible identificar singularmente las conexiones a sus terminales.

No hay límite para el número de niveles de jerarquía en la descripción ABEL. Así, es factible construir un multiplexor 8 a 1 utilizando dos multiplexores 4 a 1 y un multiplexor 2 a 1. La descripción ABEL de un multiplexor 8 a 1 de este tipo se ilustra en la figura 15, y el diagrama esquemático correspondiente, en la figura 16. Si el multiplexor 8 a 1 en la figura 15 se utilizara en otra especificación ABEL, éste también deberá tener una declaración de interfaz.

```

module 4to1mux
interface [D3 .. D0], [S1 .. S0]->Y;
Title 'multiplexor 4 a 1'
  "un multiplexor 4 a 1 construido utilizando tres multiplexores 2 a 1
Declarations

  [D3 .. D0] PIN;
  [S3 .. S0] PIN;
  Y PIN istype 'com';

  2to1mux interface (D1, D0, S -> Y); "declara la interfaz para el 2to1mux
  mux2, mux1, mux0 functional_block 2to1mux "declare 3 casos
  tmp1, tmp0 node istype 'com'; "señales internas

Equations
  mux2.D0 = D0; "S0 selecciona datos del mux 2 y el mux1 para las entradas al
  mux2.D1 = D1; "tercer multiplexor, mux0
  mux2.S = S0;
  tmp0 = mux2.Y;
  mux1.D0 = D2;
  mux1.D1 = D3;
  mux1.S = S0;
  tmp1 = mux1.Y;
  mux0.D0 = tmp0;
  mux0.D1 = tmp1;
  mux0.S = S1;
  Y = mux0.Y;
end 4to1mux

```

Figura 13. Descripción ABEL de un multiplexor 4 a 1 utilizando tres multiplexores 2 a 1.

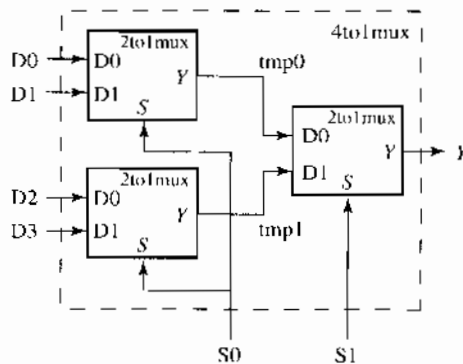


Figura 14. Diagrama esquemático que corresponde a la descripción ABEL del multiplexor 4 a 1 de la figura 13.

La descripción ABEL en las figuras 13 y 15 describen la estructura de un sistema, debido a éstas instancias de bloques funcionales, y especifican las conexiones entre ellos. En otras palabras, existe una correspondencia uno a uno entre la descripción ABEL y el diagrama esquemático correspondiente (compare las figuras 13 y 14 y las figuras 15 y 16). Como se esperaba, estos tipos de descripciones ABEL reciben el nombre de *descripciones estructurales*.

```

module 8to1mux
Title ' multiplexor 8 a 1'
'un multiplexor 8 a 1 construido utilizando dos multiplexores 4 a 1 y un multiplexor 2 a 1.
Declarations

[D7 .. D0] PIN;
[S2 .. S0] PIN;
Y PIN istype 'com';

2to1mux interface (D1, D0, S -> Y); "declara la interfaz para el 2to1mux
mux21 functional_block 2to1mux;
4to1mux interface ([D3 .. D0], [S1 .. S0] -> Y); "interfaz para el 4to1mux
mux411, mux410 functional_block 4to1mux;
tmp1, tmp0 node istype 'com'; "señales internas

Equations
mux410.D0 = D0;
mux410.D1 = D1;
mux410.D2 = D2;
mux410.D3 = D3;
mux410.S1 = S1;
mux410.S0 = S0;
tmp0 = mux410.Y;
mux411.D0 = D4;
mux411.D1 = D5;
mux411.D2 = D6;
mux411.D3 = D7;
mux411.S1 = S1;
mux411.S0 = S0;
tmp1 = mux411.Y;
mux21.D0 = tmp0;
mux21.D1 = tmp1;
mux21.S = S2;
Y = mux21.Y;
end 8to1mux

```

Figura 15. Niveles múltiples de jerarquía en una descripción ABEL de un multiplexor 8 a 1.

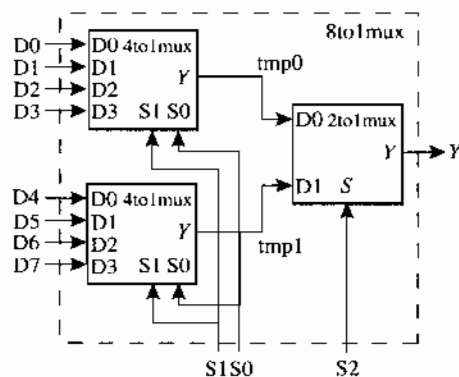


Figura 16. Diagrama esquemático que corresponde a la descripción ABEL de la figura 15.

```

module sumador
interface (A, B, Cin -> S, Cout);
Title 'celda de sumador completo'
Declarations
A PIN;
B PIN;
Cin PIN;
S PIN istype 'com';
Cout PIN istype 'com';
Equations
TRUTH_TABLE
([A, B, Cin] -> [S, Cout])
[0,0,0] -> [0,0];
[0,0,1] -> [1,0];
[0,1,0] -> [1,0];
[0,1,1] -> [0,1];
[1,0,0] -> [1,0];
[1,0,1] -> [0,1];
[1,1,0] -> [0,1];
[1,1,1] -> [1,1];
end adder

```

a)

```

module sh_reg
interface (input, clock -> output);
Title 'registro de corrimiento'

Declarations
clock, input PIN;
output PIN istype 'com'; 'reg'; "Q3 is MSB
Q3 .. Q0 NODE istype

Equations
[Q3 .. Q0].clk = clock;
output = Q0.fb;
Q0 := Q1.fb;
Q1 := Q2.fb;
Q2 := Q3.fb;
Q3 := input;
end sh_reg

```

b)

```

module serial_add
Title 'sumador en serie de 4 bits'
Declarations
clock, data PIN;
S PIN istype 'com';
carry NODE istype 'reg';
adder interface (A, B, Cin -> S, Cout);
sh_reg interface (input, clock -> output);
adder function_block adder;
sh_reg function_block sh_reg;
Equations
carry.clk = clock;
adder.A = data;
adder.B = sh_reg.output;
sh_reg.input = adder.S;
sh_reg.clock = clock;
carry := adder.Cout;
adder.Cin = carry.fb;
end serial_add

```

c)

Figura 17. Descripción ABEL para el ejemplo 1.

EJEMPLO 1

Escriba la descripción ABEL de un sumador en serie de 4 bits. Suponga que se almacena un operando en el registro de corrimiento y que el resultado se almacena en este mismo registro. Escriba la descripción jerárquicamente creando un módulo de registro de corrimiento y un módulo de sumador e inclúyalos en el módulo del sumador en serie.

El sumador es un circuito pequeño, por lo que resulta conveniente escribir una descripción de tabla de verdad, como en la descripción ABEL de la figura 17a. Confirme todos los detalles línea por línea. El registro de corrimiento en la figura 17b debe correr hacia la derecha al menos. Debe contar con una entrada externa para la posición del bit más significativo y la salida debe ser la posición del bit menos significativo. La única entrada además de los datos es el reloj. (Verifique cada línea.) ■

El sumador en serie de 4 bits puede construirse ahora instanciando el sumador y el registro de corrimiento como en la figura 17c. Aparte de eso sólo se requiere un flip-flop para almacenar la salida del acarreo del sumador entre los ciclos de reloj. Dibuje un diagrama de bloque que muestre las interconexiones de flip-flop con cada una de las dos unidades con sus propias líneas de entrada y salida.

EJEMPLO 2

Un *registro de corriente de barril* es un bloque lógico combinatorio que desplaza una palabra de datos un número seleccionado de posiciones de bit con base en una entrada de control. Escriba una descripción ABEL para un registro de corrimiento de barril de 8 bits con tres entradas de control. La palabra de entrada debe desplazarse a la derecha (del bit más significativo al menos significativo) introduciendo ceros en las posiciones vacías.

Necesitamos una sentencia *module* y una sentencia *end* que marquen el inicio y la terminación de esta descripción ABEL. Podemos nombrar el módulo de cualquier manera, aunque es mejor elegir un nombre descriptivo, como *shifter*. El título puede ser más descriptivo y debe ser suficiente para resumir la función del módulo.

El corrimiento deseado tiene ocho entradas de datos, ocho salidas de datos y tres entradas de control. Las señales de entrada de datos se relacionan lógicamente, por lo que deben especificarse como un bus. Es necesario especificar de manera similar las salidas de datos y las entradas de control. ¿Qué método de la descripción ABEL debe elegirse, tabla de verdad o ecuaciones? Imagine cuántos renglones tendría una tabla de verdad. Es evidente, entonces, que además de ser inapropiada por esa razón, una descripción ABEL de tabla de verdad reproducida aquí ¡añadiría 10% al costo del libro! ¿La descripción utilizando ecuaciones debe ser operacional —esto es, con ecuaciones lógicas escritas para las salidas— o de comportamiento? ¡Lo primero implicaría un trabajo gigantesco! La cuestión obvia que debe realizarse es escribir una descripción de comportamiento. Cuando la entrada de control tiene el valor numérico j ($0 \leq j \leq 7$), la salida es la entrada desplazada j posiciones a la derecha. Aunque la descripción ABEL se indica en la figura 18, escribala usted mismo antes de ver la solución. ■

Antes de continuar con un desarrollo adicional de ABEL, nos ocuparemos brevemente de algunos dispositivos que es posible utilizar para implementar circuitos usando los resultados de una descripción ABEL.

2 DISPOSITIVOS LÓGICOS PROGRAMABLES (PLD)

Las arquitecturas de los dispositivos del arreglo lógico programable (PLA) y de la lógica del arreglo programable (PAL) se presentaron en el capítulo 4. Repase esa sección si es necesario. Los dispositivos lógicos programables se emplean ampliamente en la práctica, y ABEL es en especial útil en la formulación de diseños que los utilizan. Esta sección es sumamente descriptiva. No lea únicamente la descripción de cada figura sino examine los detalles de cada una cuando se explique.

```

module corrimiento
Title 'corrimiento de barril derecho de 8 bits'
Declarations
[in7 .. in0], [c2 .. c0] PIN; "in7 is the MSB
[out7 .. out0] PIN istype 'com'; "out7 is the MSB
in = [in7 .. in0];
c = [c2 .. c0];
out = [out7 .. out0];
Equations
when c==0 then "no hay corrimiento
    Out = in;
else when c==1 then
    [out7 .. out0] = [0, in7 .. in1];
else when c==2 then
    [out7 .. out0] = [0, 0, in7 .. in2];
else when c==3 then
    [out7 .. out0] = [0, 0, 0, in7 .. in3];
else when c==4 then
    [out7 .. out0] = [0, 0, 0, 0, in7 .. in4];
else when c==5 then
    [out7 .. out0] = [0, 0, 0, 0, 0, in7 .. in5];
else when c==6 then
    [out7 .. out0] = [0, 0, 0, 0, 0, 0, in7 .. in6];
else when c==7 then
    [out7 .. out0] = [0, 0, 0, 0, 0, 0, 0, in7];
end shifter

```

Figura 18. Descripción ABEL del registro de corrimiento de barril.

La configuración del PAL16L8 se muestra en la figura 19. (Se presenta aquí debido no a su valor intrínseco, sino como introducción a los PLD de utilidad más general que se presentan más adelante.) Se trata de un PAL convencional con 64 compuertas AND que tiene 32 entradas (16 variables y sus complementos). Cuenta con un máximo de 8 salidas combinatorias, 10 entradas dedicadas y 6 terminales o bidireccionales que es posible programar para que sean entradas o salidas. (Examine la figura para confirmar todo esto.) Los búfers que accionan las salidas bidireccionales son de tres estados con control programable. Puede programarse una terminal bidireccional de E/S como una entrada deshabilitando el búfer de salida o como una salida habilitándolo. Es factible configurar este dispositivo para implementar circuitos multinivel, o incluso circuitos secuenciales asíncronos utilizando una de las salidas como una entrada al arreglo AND. Las patillas de E/S se retroalimentan al arreglo AND, por lo que pueden programarse como entradas para las mismas salidas o para diferentes.

El PAL16L8 se incorporó en el diseño de circuitos cuando la tecnología bipolar era de uso común. Puesto que es fácil obtener mayores escalas de integración en la tecnología CMOS, la transición de la industria a CMOS ha reducido la utilidad de la arquitectura PAL16L8. Su sustituto en la tecnología CMOS es el PALCE16V8; su diagrama esquemático lógico se muestra en la figura 20. El arreglo AND en la figura 20 es exactamente el mismo que el de la figura 19.

Advierta, sin embargo, las diferencias entre el circuito de salida de ambos. La salida del PAL16L8 consiste en un arreglo de compuertas OR y de inversores de tres estados.

Una lectura cuidadosa de la figura 20 hará claro que la salida del PALCE16V8 difícilmente puede denominarse un arreglo de meras compuertas; en vez de eso, se le denomina un arreglo de *macrocelas* (*macro* debido a que las celdas son bastante más grandes). La macrocelda de salida en este dispositivo PAL es flexible —de ahí la *V* (en inglés *versatile*) en su nombre. Un diagrama esquemático detallado de la macrocelda de salida se presenta en la figura 21. Los tamaños de transistor en la tecnología CMOS (MOSFET) disminuyen al mismo paso que la miniaturización

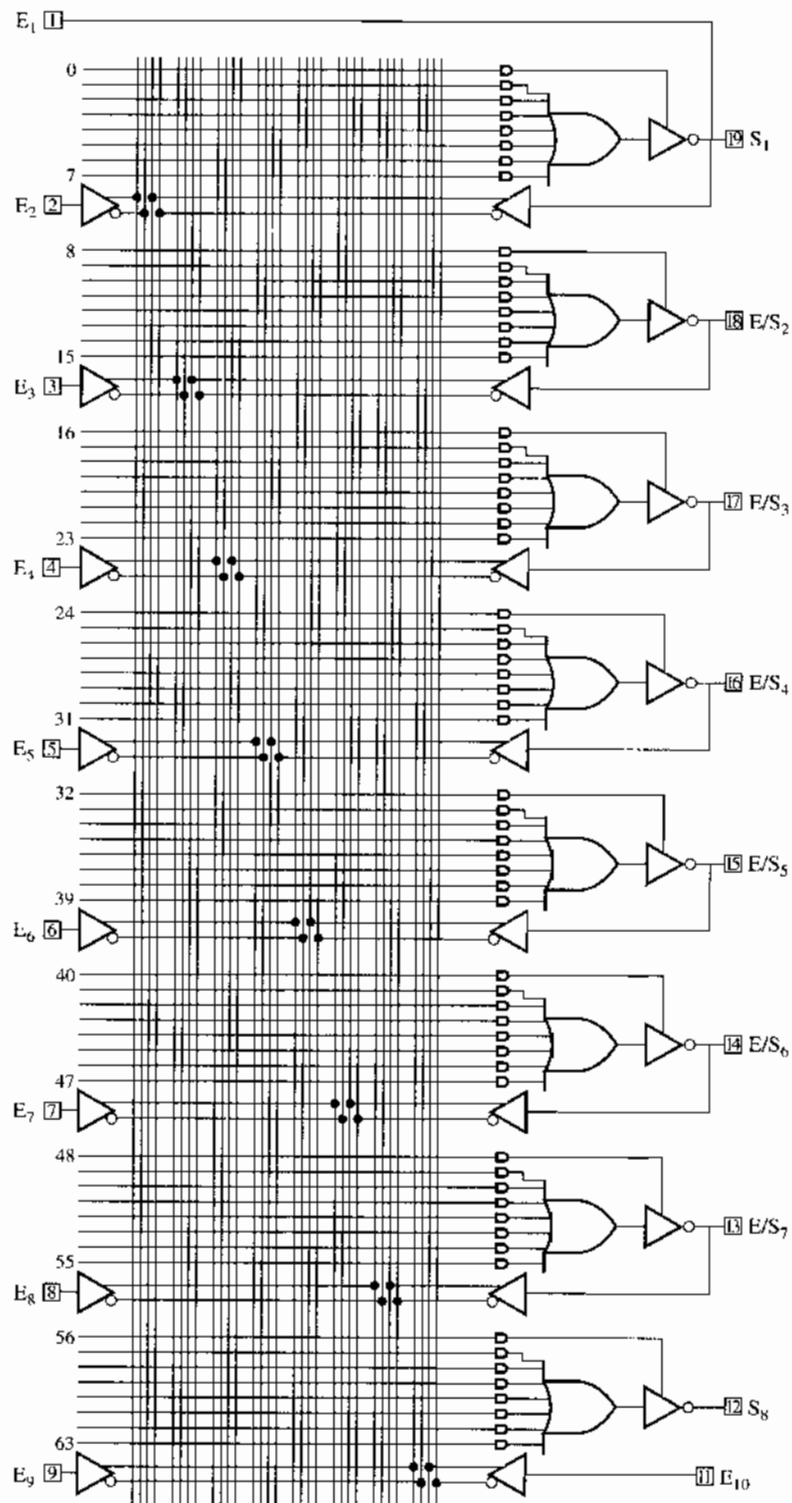


Figura 19. Diagrama esquemático lógico del PAL16L8. (Cortesía de Texas Instruments.)

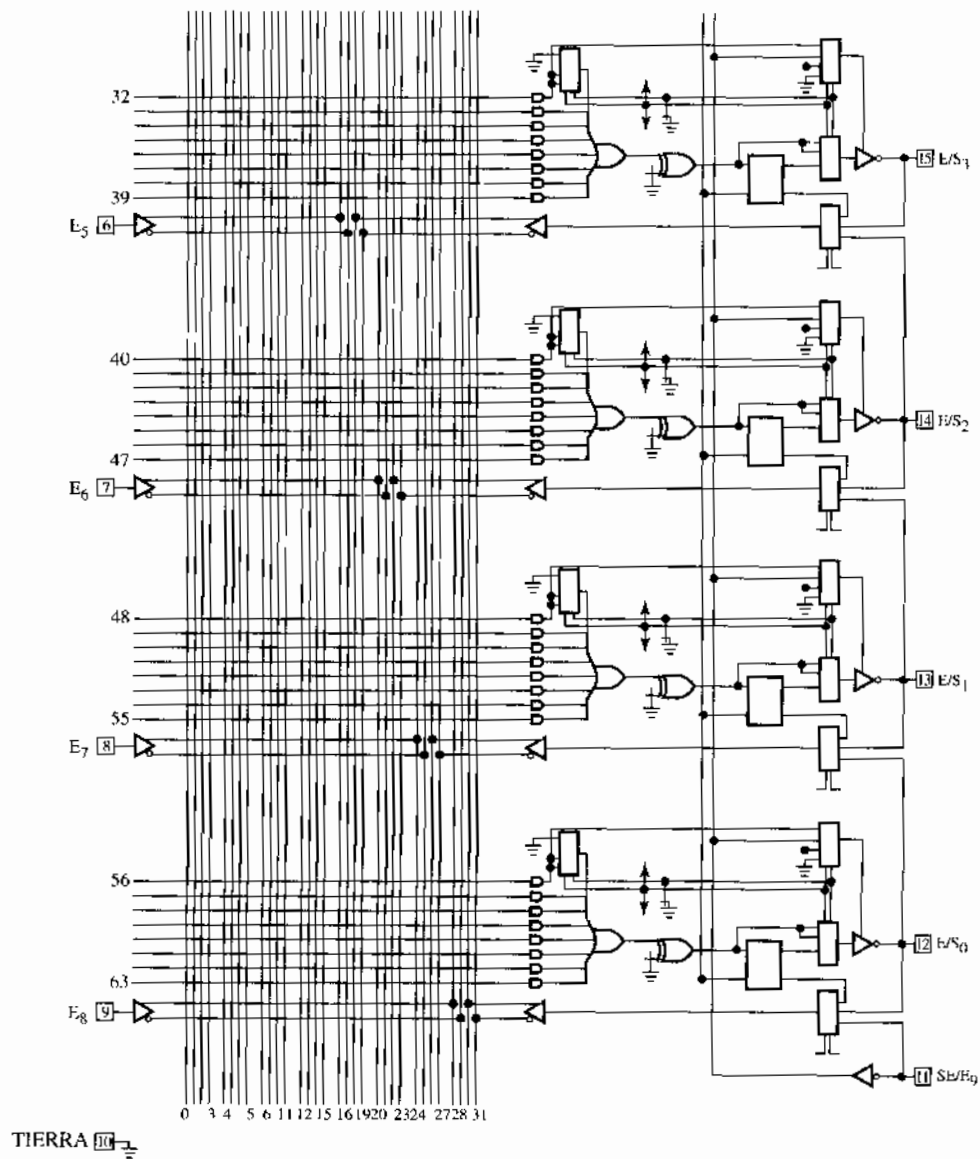


Figura 20. Diagrama esquemático lógico del PALCE16V8. (Diagrama proporcionado por Lattice Semiconductor Corporation.)

de los elementos de los circuitos integrados como las trazas de aluminio. Los avances en la tecnología CMOS permiten incrementar la complejidad del sistema en un solo circuito integrado con poco (si es que hay alguno) incremento en el costo.

En la figura 22 se presentan varias configuraciones de la macrocelda de salida. Si se usa la configuración indicada en la figura 22c, entonces el comportamiento del PALCE16V8 es idéntico al del PAL16L8.

El PALCE16V8 tiene una utilidad bastante mayor debido a la existencia de un flip-flop a la salida de cada compuerta OR. Con el PAL16L8, en cambio, resulta necesario utilizar circuitos integrados de flip-flop separados (por ejemplo, el 7474s) para implementar los circuitos secuenciales síncronos. La mayor parte de los circuitos síncronos que se diseñaron en el capítulo 6 pueden

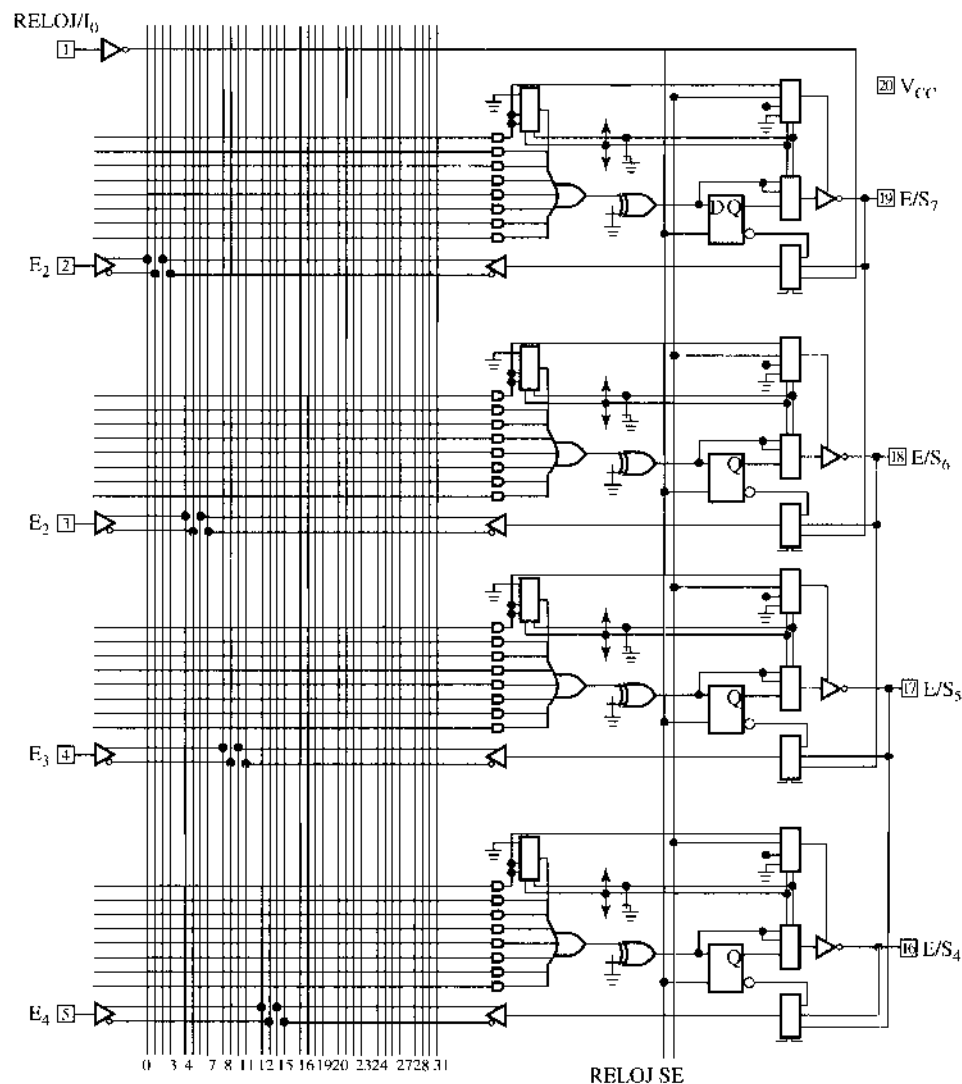


Figura 20. (Continuación.)

implementarse con un PALCE16V8, aunque podrían requerirse tres o más circuitos integrados si se usa el PAL16L8.

Ejercicio 5. Describa cómo podría implementar un cerrojo *SR* con el PAL16L8.

Ejercicio 6. Describa cómo implementar un circuito con cuatro niveles lógicos utilizando el PALCE16V8. ¿Es posible implementar también circuitos con un número impar de niveles? Si no, ¿de qué manera se implementan este tipo de circuitos? ♦

Dispositivos lógicos programables complejos

Los dispositivos lógicos programables que se describieron en la sección anterior parecen ser bastante intrincados, aunque su complejidad palidece en comparación con la de las interconexiones

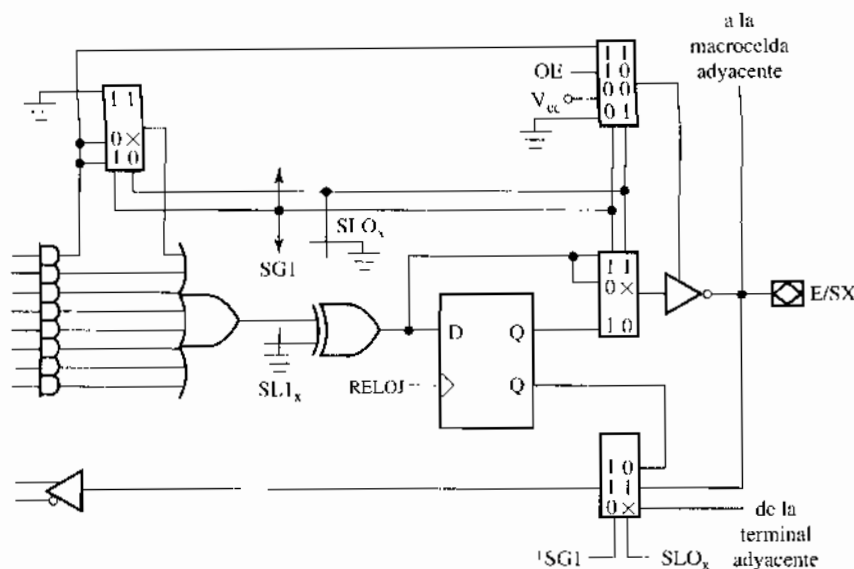


Figura 21. Diagrama esquemático lógico de la macrocelda de salida PALCE16V8. (Diagrama proporcionado por Lattice Semiconductor Corporation.)

de los dispositivos a los que se les da el nombre de *dispositivos lógicos programables complejos* (CPLD).⁷ Tales dispositivos contienen varias arquitecturas PLD convencionales, como la PALCE16V8, con un arreglo de interconexiones programables llamado a menudo *matriz de interconexión*. La arquitectura de uno de éstos, la familia de CPLD Xilinx XC9500, se presenta en la figura 23. Cada uno de sus bloques de funciones (que contiene 18 macroceldas) tiene el doble de la capacidad del PALCE16V8 completo.

La arquitectura del bloque de funciones XC9500 se muestra en la figura 24. La macrocelda de un bloque de funciones del XC9500 se presenta en la figura 25 y es similar en capacidad a una macrocelda del PALCE16V8 de la figura 21.

La familia XC9500 incluye varios dispositivos; los detalles de las características de seis de ellos se resumen en la tabla de la figura 26. El más pequeño, el XC9536, es equivalente en potencia a aproximadamente 4.5 dispositivos PALCE16V8; el más grande, el XC95288, es casi 36 veces más poderoso que el PALCE16V8.⁸

Advierta en la figura 26 que el retardo de propagación característico aumenta con la complejidad del dispositivo. La arquitectura modular sugiere que son similares los factores de carga de salida de las compuertas en los dispositivos de todos los tamaños. ¿Por qué entonces se incrementa el retardo de propagación con la complejidad del dispositivo? La respuesta es que para mantener la flexibilidad en la programación del dispositivo, la complejidad de la matriz de conmutación crece de manera significativa con el número de bloques funcionales. Y es el retardo a través de dicha matriz lo que ocasiona el aumento.

⁷ En el tiempo de la publicación, varios vendedores los producían.

⁸ Durante la elaboración del manuscrito, el costo de un PALCE16V8 (retardo de 7 ns) en cantidades de 100 o más era de 2.00 dólares; el del correspondiente al XC9536 era de 3.70 dólares por dispositivo. De tal modo, el XC9536 es 4.5 veces más poderoso y sólo 1.85 veces más caro. La potencia del circuito integrado se sustenta en el hecho de que la densidad del dispositivo (la cantidad de circuitería por área unitaria) ha crecido de manera exponencial en tanto que su costo se ha reducido más o menos de manera constante por más de un cuarto de siglo.

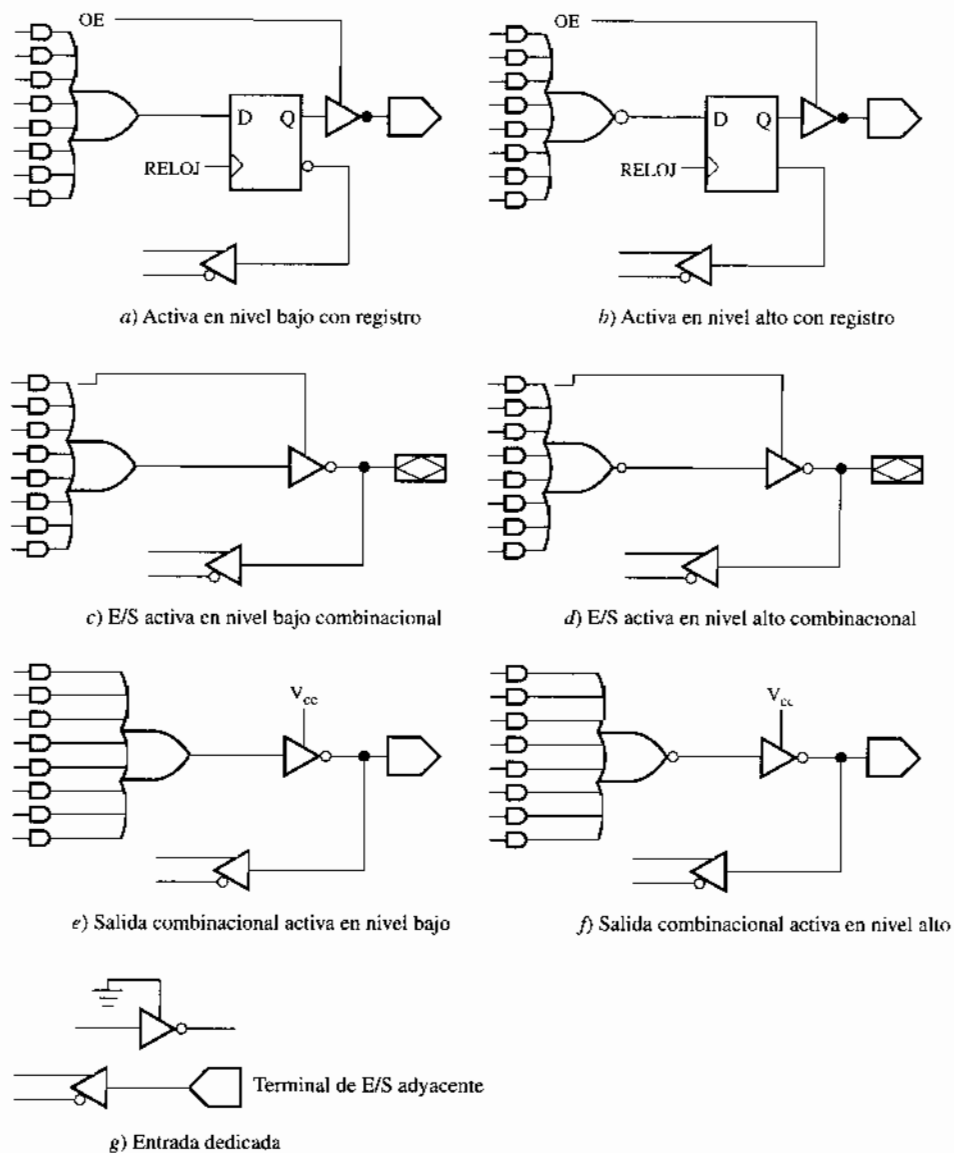


Figura 22. Diversas configuraciones de la macrocelda de salida PALCE16V8. (Diagrama proporcionado por Lattice Semiconductor Corporation.)

El costo no se incluye en las especificaciones de la figura 26 debido a que cambia rápidamente en el tiempo. El costo por tamaño unitario de un dispositivo es más o menos constante. Por ello es que el costo de los dispositivos de la figura 26 aumenta de izquierda a derecha casi en proporción con el número de compuertas utilizables. La diferencia de costo entre un PALCE16V8 y un XC9536 no es tan sustancial como la correspondiente entre un XC9536 y un XC95144, dado que la familia XC9500 se fabrica con una tecnología más avanzada que la PALCE16V8. A medida que avanza la tecnología, es posible fabricar más compuertas en un área de tamaño unitario.

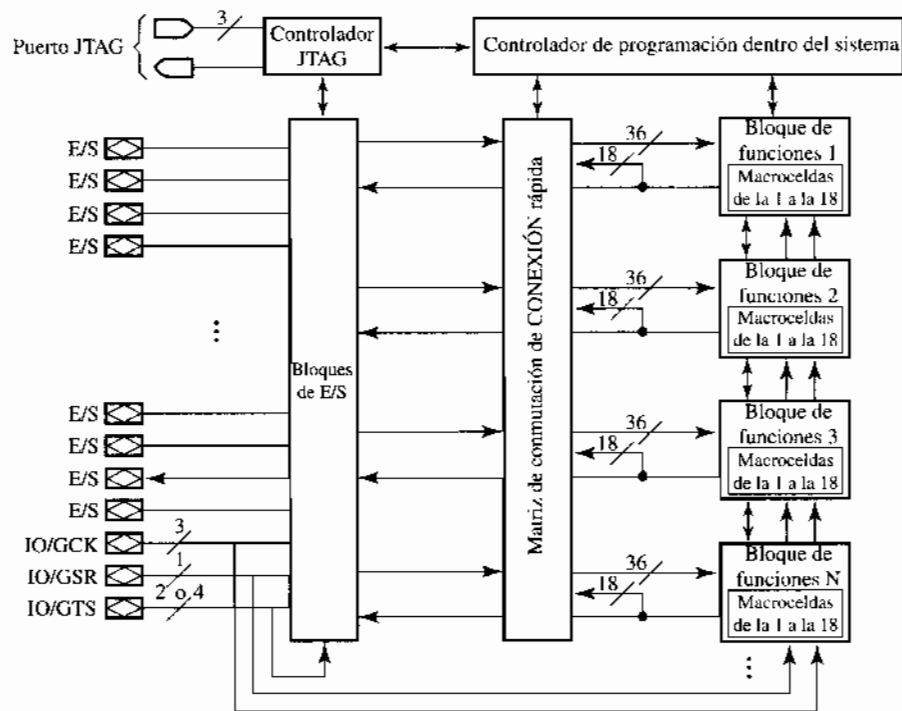


Figura 23. Arquitectura de la familia CPLD XC9500 de Xilinx. (Figura basada o adaptada de figuras y textos propios de Xilinx, Inc., cortesía de Xilinx, Inc. © Xilinx, Inc. 1999. Derechos reservados.)

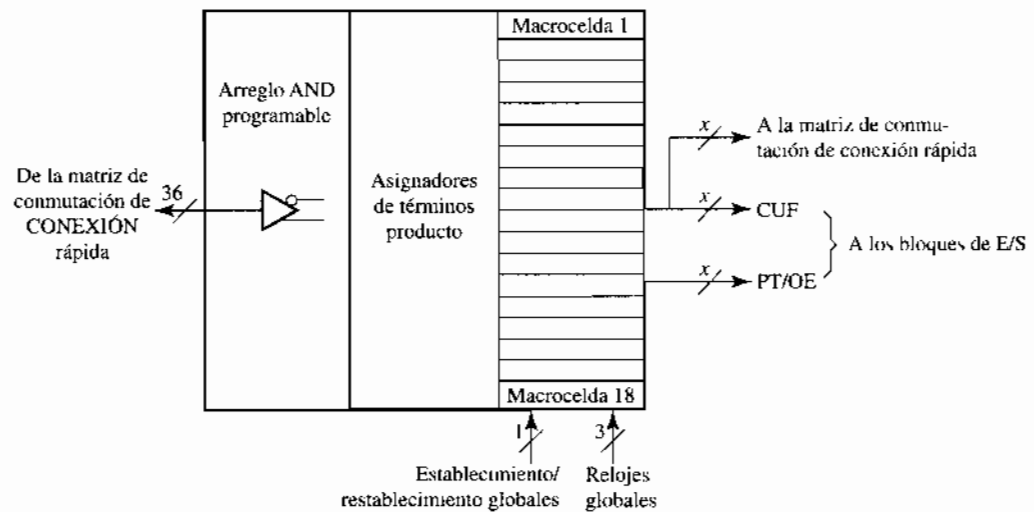


Figura 24. Arquitectura del bloque de funciones del PLD XC9500. (Figura basada o adaptada de figuras y textos propios de Xilinx, Inc., cortesía de Xilinx, Inc. © Xilinx, Inc. 1999. Derechos reservados.)

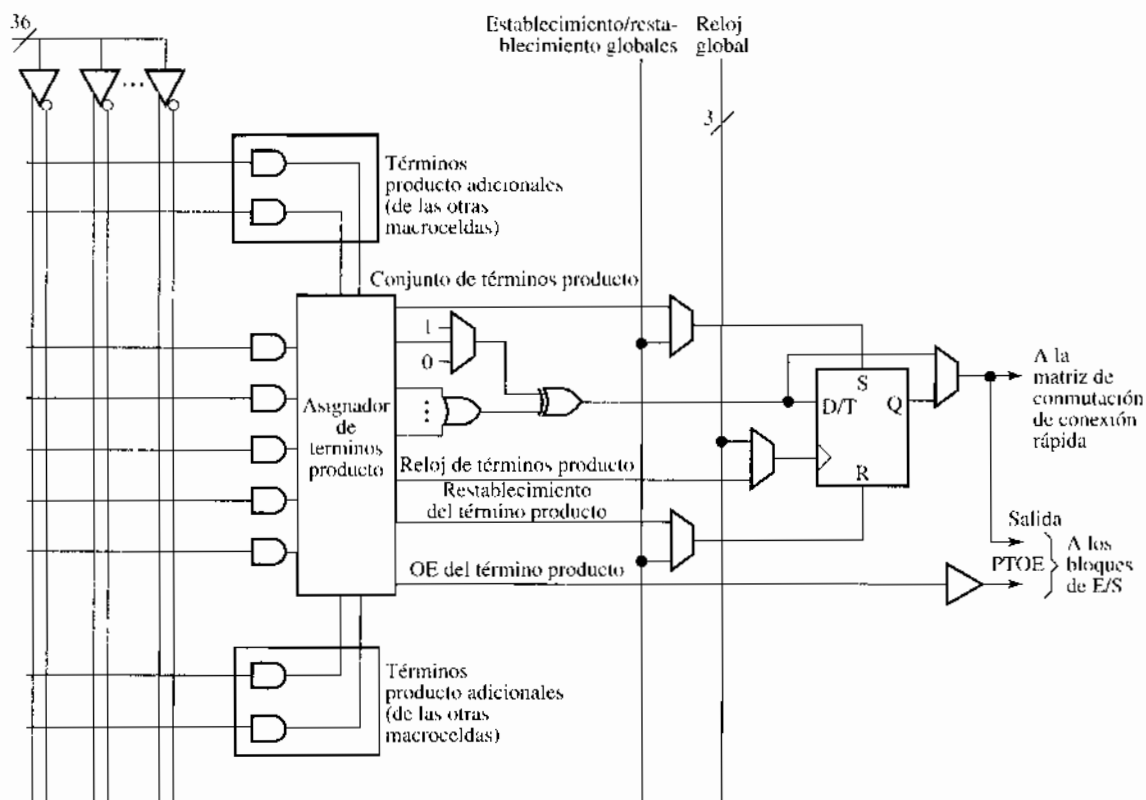


Figura 25. Arquitectura de macroceldas de un bloque de funciones XC9500. (Figura basada o adaptada de figuras y textos propios de Xilinx, Inc., cortesía de Xilinx, Inc. © Xilinx, Inc. 1999. Derechos reservados.)

	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
Macrocelas	36	72	108	144	216	288
Compuertas utilizables	800	1,600	2,400	3,200	4,800	6,400
Registros	36	72	108	144	216	288
t_{pd} (ns)	5	7.5	7.5	7.5	10	15
t_{su} (ns)	4.5	5.5	5.5	5.5	6.5	8.0
t_{cu} (ns)	4.5	5.5	5.5	5.5	6.5	8.0
f_{CNT} (MHz)	100	125	125	125	111	195
$f_{SYSTEMA}$ (MHz)	100	183	183	183	167	156

Figura 26. Especificaciones de dispositivo para la familia XC9500 de Xilinx. (Figura basada o adaptada de figuras y textos propios de Xilinx, Inc., cortesía de Xilinx, Inc. © Xilinx, Inc. 1999. Derechos reservados.)

Ejercicio 7. El XC95144 contiene cuatro veces más compuertas utilizables que el XC9536. Si éstos se hubieran fabricado con la misma tecnología, ¿qué diferencia de precio esperarías entre ellos? Si el XC95144 se hubiera fabricado con una tecnología más avanzada en la cual pudieran fabricarse dos veces más compuertas utilizables en el mismo espacio que antes, ¿qué diferencia de costo esperarías?

Arreglos de compuertas programable en campo

Un PLD que no se ha explicado es el arreglo de compuertas programable en campo (FPGA). Los FPGA son dispositivos programables que difieren de los PLD en su complejidad y organización; contienen circuitería capaz de implementar diseños con hasta decenas de miles de compuertas; se organizan como arreglos bidimensionales que interconectan bloques lógicos.

En la figura 27 se presenta la organización de un FPGA Xilinx característico. Éste contiene un arreglo de bloques lógicos configurables (BLC) con interconexiones programables entre ellos. La estructura de un BLC se muestra en la figura 28. Éste se conoce como un FPGA basado en RAM puesto que, en cada uno de los BLC, el dispositivo se programa escribiendo bits en la RAM.

Xilinx, y otros vendedores, fabrican una serie de FPGA con complejidad variable. En la figura 29 se presenta un resumen de su serie de dispositivos XC4000. Puesto que es un FPGA basado en RAM, es posible configurar los BLC como memoria direccionable de lectura/escritura. Lo anterior hace que esta tecnología particular sea eficiente para la implementación de búfers y colas.

Cada uno de los numerosos vendedores de FPGA tiene su propia arquitectura y configuración de bloques lógicos. Los vendedores usan varias tecnologías para programar los dispositivos. Los dispositivos Xilinx pueden reconfigurarse un número arbitrario de veces, aunque es necesario un dispositivo acompañante, como una EPROM, para el almacenamiento permanente de la configuración.

El sistema se diseña de manera que la EPROM cargue la configuración en el FPGA luego del arranque. Algunos fabricantes utilizan tecnologías que se emplean en EPROM o PLA. Estos dispositivos quizá sean programables una sola vez, o es posible que se borren un número limitado de veces, del orden de cientos o miles.

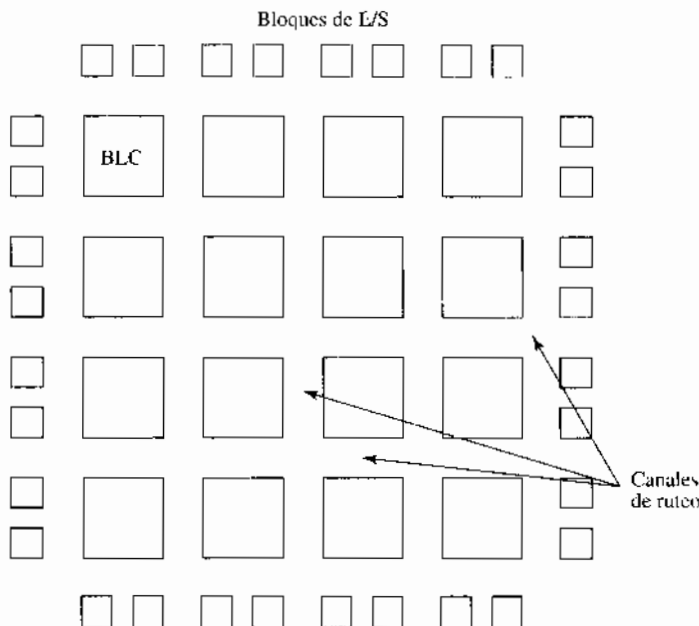


Figura 27. Arquitectura de FPGA de Xilinx.

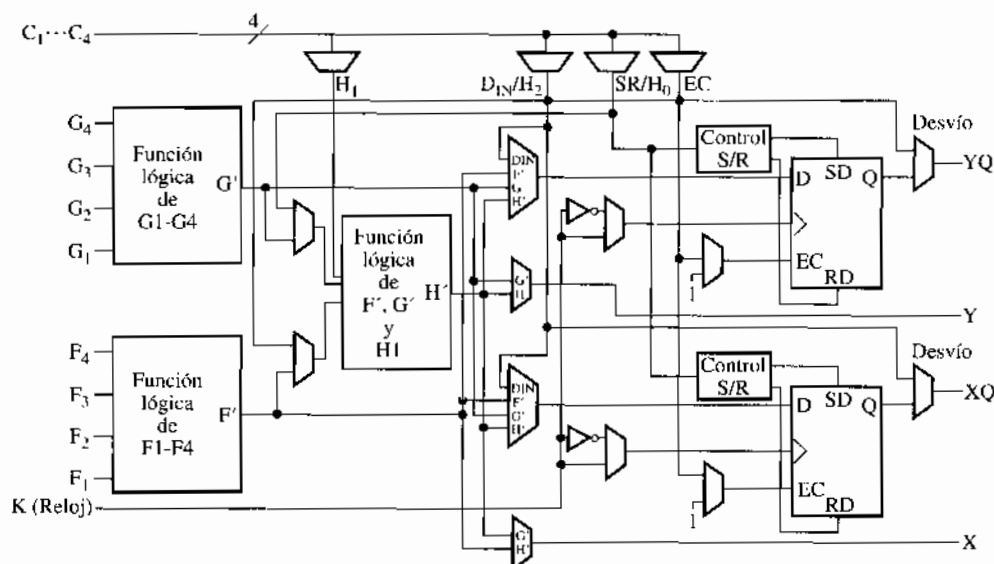


Figura 28. Bloque lógico configurable (BLC) de los FPGA de Xilinx. (Figura basada o adaptada de figuras y textos propios de Xilinx, Inc., cortesía de Xilinx, Inc. © Xilinx, Inc. 1999. Derechos reservados.)

Dispositivo	Celdas lógicas	Compuertas lógicas máx. (RAM núm.)	Bits RAM máx. (Lógica núm.)	Rango característico de compuertas de compuertas (lógicas y RAM)*	Matriz BLC	Número de BLC totales	Número de flip-flops	E/S máx. de usuario
XC4002XL	152	1,600	2,048	1,000-3,000	8 × 8	64	256	64
XC4003E	238	3,000	3,200	2,000-5,000	10 × 10	100	360	80
XC4005E/XL	466	5,000	6,272	3,000-9,000	14 × 14	196	616	112
XC4006E	608	6,000	8,192	4,000-12,000	16 × 16	256	768	128
XC4008E	770	8,000	10,368	6,000-15,000	18 × 18	324	936	144
XC4010E/XL	950	10,000	12,800	7,000-20,000	20 × 20	400	1,120	160
XC4013E/XL	1,368	13,000	18,432	10,000-30,000	24 × 24	576	1,536	192
XC4020E/XL	1,862	20,000	25,088	13,000-40,000	28 × 28	784	2,016	224
XC4025E	2,432	25,000	32,768	15,000-45,000	32 × 32	1,024	2,560	256
XC4028EX/XL	2,432	28,000	32,768	18,000-50,000	32 × 32	1,024	2,560	256
XC4036EX/XL	3,078	36,000	41,472	22,000-65,000	36 × 36	1,296	3,168	288
XC4044XL	3,800	44,000	51,200	27,000-80,000	40 × 40	1,600	3,840	320
XC4052XL	4,598	52,000	61,952	33,000-100,000	44 × 44	1,936	4,576	352
XC4062XL	5,472	62,000	73,728	40,000-130,000	48 × 48	2,304	5,376	384
XC4085XL	7,448	85,000	100,352	55,000-180,000	56 × 56	3,136	7,168	448

*Los valores máximos de rangos característicos de compuertas incluyen de 20-30% de los BLC utilizados como RAM.

Figura 29. Resumen de la familia XC4000 de FPGA. (Figura basada o adaptada de figuras y textos propios de Xilinx, Inc., cortesía de Xilinx, Inc. © Xilinx, Inc. 1999. Derechos reservados.)

Mientras se escribía este libro, se llevaba a cabo cierto debate acerca del uso de los FPGA contra los ASIC (circuitos integrados de aplicación específica) en productos manufacturados.⁹ Los FPGA se usan principalmente en el desarrollo de prototipos de productos y por lo común se

⁹ Véase el capítulo 2 para una breve explicación de los ASIC.

sustituyen por ASIC cuando el producto entra en la etapa de producción. Debido a que el FPGA es genérico, las compuertas por lo común no se utilizan de manera adecuada. Los ASIC son más pequeños debido a que sólo se incluyen en el circuito integrado aquellas compuertas que son necesarias; éstas se arreglan en la CI de modo que se minimice el área total. Lo anterior permite que el costo por componente de un ASIC sea menor que el de un FPGA. Debido a que las compuertas en un FPGA se sustituyen en localidades fijas y la estructura se diseña para acomodar circuitos arbitrarios, resulta más difícil utilizar eficientemente las compuertas. Por esta razón, los ASIC tienden a tener menor retardo que los FPGA y pueden operarse con un reloj del sistema más rápido.

Sin embargo, como se explica en el capítulo 3, los ASIC presentan grandes costos de desarrollo asociados (lo que se denomina cargos de ingeniería no recurrente, o NRE). El gran cargo de NRE para una ASIC proviene del esfuerzo de ingeniería necesario para diseñar el circuito integrado y el herramental (en la forma de máscaras fotolitográficas) necesario para fabricarlo. Es por ello que los ASIC tal vez no se usen en productos que se venden en bajo volumen.

La decisión de usar un FPGA o un ASIC en un producto por lo general es económica y pondera el costo de desarrollar el ASIC en contra del costo por componente del ASIC respecto al FPGA. Si se espera que el producto se venda en gran volumen (esto es, bipers o teléfonos celulares), la inversión que se requiere entonces para crear un ASIC se recuperará varias veces más con los ahorros en los costos por parte del ASIC contra el FPGA. Sin embargo, sigue teniendo sentido utilizar los FPGA para diseñar un prototipo de un producto, ya que se aprende mucho durante el proceso de elaboración del prototipo y, en consecuencia, quizás sean necesarios cambios subsecuentes en el diseño. No tendría sentido económico utilizar un ASIC antes de haber terminado el producto.

3 EL FLUJO DE DISEÑO PARA LAS ESPECIFICACIONES HDL

Equipados con conocimiento del ABEL como un lenguaje de descripción de hardware y de una comprensión general de ciertos CPLD, estamos listos para emprender el proceso de diseño utilizando estos elementos. Dada una especificación de diseño digital, el proceso de diseño consiste en una secuencia de varios pasos distintos. El flujo que ocurre de un paso a otro durante el proceso se conoce como el *flujo de diseño*. Éste emplea varias herramientas diferentes de diseño asistido por computadora, cada una de las cuales opera sobre la especificación que hace posible arribar al sistema físico. A fin de diseñar de manera efectiva con el uso de cualquier HDL, incluyendo ABEL, es necesario entender estas herramientas de diseño asistido por computadora y sus métodos de operación. Esta sección describe algunas de estas herramientas.

Antes de que se efectúe cualquier implementación de hardware, necesitamos asegurar que la descripción ABEL —producida con tanto esfuerzo— sea correcta. Para verificar la descripción ABEL, una simulación sería lo más apropiado. Es posible obtener comercialmente este tipo de herramientas de simulación.¹⁰

La herramienta de síntesis traduce la especificación ABEL en una que es independiente del dispositivo y del nivel de la compuerta. Después la herramienta de mapeo a tecnología traduce la descripción del nivel de compuerta a un formato que se utiliza para programar un dispositivo físico específico (por ejemplo, un PALCE16V8). ¿Por qué no pasar directamente de la especificación ABEL al diseño utilizando dispositivos específicos?¹¹

¹⁰ Una herramienta de simulación apropiada es la que se distribuye con el software Webpack de Xilinx y puede encontrarse en el sitio Web de esta misma empresa. La dirección del sitio es (dato del tiempo en que se escribió este libro): <http://www.xilinx.com/sxpresso/webpack.htm>

¹¹ Una especificación independiente del dispositivo se efectuaría primero de manera que siguiera disponible una elección de hardware específico para utilizarse después; la descripción de nivel de compuerta es transportable de un dispositivo al siguiente.

Evidentemente, la síntesis y el mapeo a tecnología no modifican el comportamiento del sistema. Luego del mapeo a tecnología, sin embargo, hay más información disponible acerca de él. Por ejemplo, a partir de la descripción que se obtuvo de la herramienta de mapeo a tecnología, es posible determinar el factor de carga de entrada, el factor de carga de salida, el número de niveles de compuerta, así como otros parámetros. Una vez que esta información se ha adquirido, podría resultar útil ejecutar otra simulación, poniendo mucha atención en los requerimientos de temporización.

El flujo de diseño para llegar a las especificaciones del hardware consta de cinco pasos:

1. Partiendo de una descripción en palabras del problema, escriba una descripción ABEL del diseño.
2. Simule la especificación ABEL utilizando un simulador ABEL.
3. Sintetice la especificación ABEL en un formato lógico intermedio, independiente de la tecnología, a nivel de compuertas.
4. Utilice la herramienta de mapeo a tecnología para traducir el formato intermedio en una especificación física (PLD, FPGA, o biblioteca estándar).
5. Simule la especificación física utilizando una lógica de nivel de compuerta y un simulador de temporización.

Síntesis y asignación de tecnología de especificaciones ABEL

La tarea de transformar una descripción ABEL en una especificación física se divide en dos: síntesis y mapeo a tecnología. La herramienta de síntesis¹² produce una representación lógica intermedia independiente de la tecnología que corresponde a la estructura lógica de la implementación física. Un formato estándar para esta representación es el formato de *intercambio de datos electrónicos* (EDIF).

Dependiendo de la tecnología que se use para la implementación, quizás tenga que modificarse la representación intermedia producida por la herramienta de síntesis; ésta es la tarea de la herramienta de mapeo a tecnología. Por ejemplo, la herramienta de síntesis tal vez calcule una implementación que utilice una compuerta AND de 10 entradas, aunque el máximo factor de carga de entrada de compuertas en la tecnología objetivo podría ser sólo de seis. La representación intermedia debe transformarse en una especificación física que pueda implementarse utilizando los dispositivos disponibles en la tecnología objetivo (por ejemplo, un XC9500).

Las herramientas de síntesis no pueden explorar todas las implementaciones posibles de un diseño para determinar la mejor. La implementación que se calcula mediante una herramienta de síntesis resulta buena respecto a ciertas características de desempeño pero no a otras. Por ejemplo, un sumador de acarreo anticipado tiene un retardo de propagación bajo, pero presenta una gran disipación de potencia y requiere más compuertas que otras implementaciones de sumador. Un sumador de acarreo en cascada es pequeño y disipa menos potencia, aunque tiene un retardo sustancial. Numerosas implementaciones de sumador se encuentran entre estos dos extremos, y la realización más adecuada para una aplicación particular quizá sea una de ellas.

Para controlar la implementación calculada por medio de una herramienta de síntesis, es necesario que los diseñadores entiendan cómo utilizar la herramienta y cómo funciona ésta, es decir, de qué manera sintetiza las realizaciones especificadas. Los diseñadores deben dirigir las herramientas hacia la implementación que buscan.

Por omisión, las herramientas de síntesis ABEL calculan una implementación lógica de dos niveles de la especificación. Esto corresponde a la implementación con el máximo paralelismo (retardo más bajo) y casi siempre el mayor número de compuertas. Para la mayor parte de las aplicaciones ésta quizá no sea la realización deseada.

¹² Véase el pie de página 10.

Circuito	@ carry	Número de términos producto	Número de compuertas	Número de niveles de compuerta
A	0	135	455	5
B	1	77	152	10
C	2	97	266	8
D	3	84	220	7
E	4	145	517	5
F	5	135	455	5

Figura 30. Características de implementaciones de sumador sintetizadas en la figura 5.

El propio ABEL cuenta con procedimientos para alterar la implementación por omisión. ABEL contiene instrucciones que se usan de manera específica para dirigir la herramienta de síntesis hacia ciertas realizaciones. Vimos una de estas instrucciones (la instrucción @carry) en la especificación de un sumador en la figura 5. Estas instrucciones a menudo se denominan *directivas*. La orden @carry controla el número de niveles de lógica en la implementación al especificar el número de niveles entre las generaciones de acarreo. Por ejemplo, @carry 1 ordena a la herramienta de síntesis generar un acarreo entre cada nivel; esto corresponde a un sumador de acarreo en cascada. @carry 4 ordena a la herramienta de síntesis generar un acarreo entre cada cuatro etapas del sumador. Lo anterior corresponde a un sumador de acarreo anticipado con anticipo en cada sección de 4 bits y acarreo en cascada entre las secciones.

Sintetizamos la especificación del sumador de 4 bits en la figura 5 con valores de @carry entre 0 y 5. En la figura 30 se muestra un resumen de las características de la implementación. @carry significa ignorar la orden @carry. Esperamos que el número de compuertas aumente y que el número de niveles de compuerta disminuya cuando el valor de @carry aumente a partir de 1, y el número de niveles de compuerta disminuya desde luego de 10 a 5. Quizás le sorprenda por qué no se ha reducido más el número de niveles de compuerta para el circuito F en la figura 30, incluso hasta dos niveles. En realidad, la herramienta de síntesis produce una representación intermedia que tiene dos niveles, aunque la tecnología objetivo (en este caso el PLD9500 de Xilinx) no soporta compuertas con un factor de carga de entrada suficientes para implementar físicamente el circuito con dos niveles de lógica. Entonces, el número mínimo de niveles de compuerta alcanzable para un sumador de acarreo anticipado de 4 bits en el Xilinx 9500 corresponde a 5.

Considere el número de compuertas en cada realización de la figura 30. La razón por la que el número de compuertas en el circuito D es más pequeño que el correspondiente al circuito C (y de modo similar para los circuitos E y F) es la manera en que se genera el acarreo dentro del sumador de 4 bits. La implementación C tiene un acarreo en cascada entre los bits 1 y 2 (el bit 0 es el menos significativo) y entre el bit 3 y la salida del acarreo. La implementación D contiene un acarreo en cascada entre los bits 2 y 3 únicamente (con acarreo anticipado de los bits 0 a 2). De tal manera, el circuito D es más pequeño (tiene menos compuertas) y más rápido que el circuito C. El circuito F es más pequeño que el circuito E por una razón similar.

Por lo general, el retardo (número de niveles de compuerta) y el tamaño (número de compuertas) se relacionan de manera inversa. El espacio de diseño puede caracterizarse mediante la gráfica que se muestra en la figura 31, donde las realizaciones de circuito se encuentran dentro de las fronteras indicadas en el espacio bidimensional. Advierta que es posible incrementar el número de compuertas para aumentar la velocidad del circuito, aunque después de un punto existe un rendimiento disminuido.

La tarea de la herramienta de síntesis es transformar la especificación HDL en un formato intermedio que en cierto sentido está optimizado (por ejemplo, número de compuertas menor). La herramienta recurre a técnicas de minimización booleana, de minimización de estados, así como

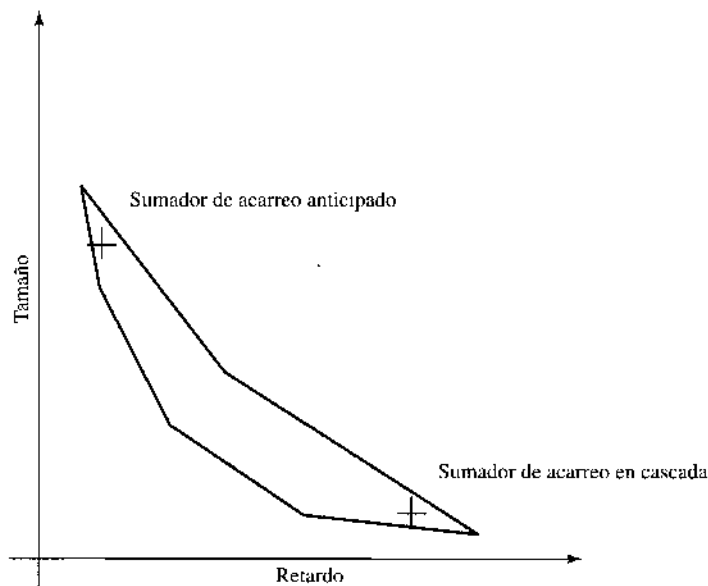


Figura 31. Espacio de implementación característico para un circuito digital.

a técnicas de minimización más avanzada para optimizar el circuito. En general, no es posible que las herramientas de síntesis produzcan resultados por completo óptimos, pues es factible que generen resultados para diseños más complejos que no es viable alcanzar por medio de métodos de diseño manuales. Además, es probable identificar una implementación con mucho mayor rapidez con la asistencia de herramientas de síntesis.

Las herramientas de mapeo a tecnología transforman la salida de una herramienta de síntesis en una forma que especifica una implementación utilizando un PLD o ASIC particular. Estas herramientas se denominan algunas veces *ajustadores*, puesto que ajustan un diseño a un componente de circuito integrado particular. En el caso de un PLD, la herramienta de mapeo a tecnología decide los términos y variables de estado del producto que debe asignar a cada macrocelda, y las entradas y salidas para cada terminal. La herramienta de mapeo a tecnología produce un archivo de computadora binario que es adecuado para descargarse hacia el PLD con el fin de configurarlo. Se utilizan software y hardware especializados (conocidos como programador de dispositivo) para descargar la configuración correspondiente a un dispositivo. Este tema no se considerará más en este libro, aunque debe llevarse a cabo en un laboratorio.

Simulación de especificaciones ABEL

Después de que se ha obtenido una especificación de hardware inicial (en nuestro caso, una descripción ABEL), ésta debe simularse para verificar su correcto funcionamiento. Dicha especificación es independiente de la tecnología, lo que quiere decir que el simulador no tiene conocimiento acerca de retardos de compuerta, tiempos de establecimiento y retención, o parásitos de interconexión.¹³ Es posible utilizar la simulación HDL para verificar la corrección fun-

¹³ Los *parásitos de interconexión* son la capacitancia y la resistencia de los alambres en un circuito integrado. Su discusión está más allá del objetivo de este libro. Sin embargo, los transistores son tan rápidos en los circuitos integrados modernos que al estimar el retardo de las trayectorias lógicas, es necesario tomar en consideración el retardo de una señal en un alambre. (El retardo de una señal de voltaje en un alambre es proporcional al producto de la resistencia y la capacitancia del alambre.)

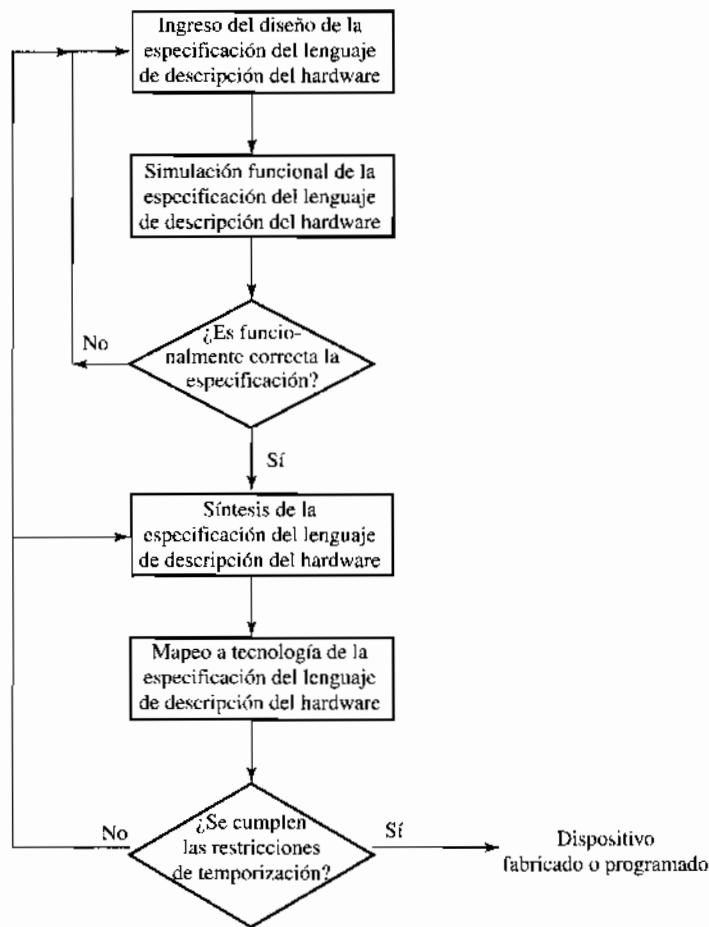


Figura 32. Diagrama de flujo del proceso de diseño con un HDL.

cional antes de sintetizar una implementación, aunque no puede utilizarse para comprobar el desempeño. Es necesario generar una secuencia de entradas de prueba en forma manual cuidadosa con el fin de verificar que una especificación es correcta para entradas arbitrarias. En sistemas grandes es imposible simular todos los patrones de entrada y estados posibles, aunque debe establecerse la certidumbre de la operación correcta.

La especificación de dispositivo producida por la herramienta de mapeo a tecnología contiene información importante del retardo tal como el correspondiente a la compuerta, los tiempos de establecimiento y retención, y el retardo del cableado. Es viable extraer un modelo de simulación de la especificación del dispositivo que es adecuada para verificar el desempeño y la temporización del diseño. Si bien una implementación quizá funcionalmente sea correcta, tal vez no cumpla con los requisitos de desempeño. De tal manera, se requiere una simulación después del mapeo a tecnología para verificar el desempeño del circuito.

Si el requisito de desempeño no se cumple mediante la implementación, la especificación debe resintetizarse con mayor énfasis en el desempeño del circuito. Este ciclo se repite hasta que se cumple el requerimiento. Si lo anterior no es posible, resulta necesario elegir una tecnología más rápida. También es factible que el circuito sintetizado sea demasiado rápido. ¿Tiene esto alguna consecuencia? Quizá se encuentre una implementación que tenga menos compuertas y que siga cumpliendo con el requerimiento de temporización. En la figura 32 se describe un resumen del flujo de diseño mediante un diagrama de flujo. Usualmente se efectúan varias iteraciones del

flujo de diseño con especificaciones iniciales diferentes (equivalentes en cuanto a funcionalidad) y parámetros de síntesis para explorar el espacio de la implementación (por ejemplo, figura 31) de un diseño. Con frecuencia cambian los requerimientos del subsistema durante el desarrollo de un sistema. Es necesario entender las limitaciones de diversos dispositivos y tecnologías respecto a una especificación.

A pesar de que ABEL es el lenguaje de descripción del hardware que se cubrió en este capítulo, casi todos los conceptos se aplican al diseño con otros HDL, incluyendo VHDL y Verilog. El flujo de diseño que se describe en la figura 32 es similar para todos los HDL. Estos pueden simularse, y están disponibles numerosos simuladores comerciales. Las herramientas de síntesis y mapeo a tecnología se obtienen también a partir de muchas compañías de software CAD, pues se trata de software para todas las tareas en el flujo de diseño.

RESUMEN Y REPASO DEL CAPÍTULO

Este capítulo utiliza un lenguaje de descripción de hardware (HDL) en el diseño de circuitos lógicos, que incluye dispositivos lógicos programables (PLD). Se caracterizan los dispositivos de lógica de arreglo programable (PAL), aunque se presentan también PLD complejos (CPLD). El lenguaje específico utilizado es el que se denomina lenguaje de expresión booleana avanzado (ABEL). Se ofrecen suficientes detalles del lenguaje para permitirle llevar a cabo este tipo de diseños. Los temas específicos incluyeron:

- Diferencia entre un HDL y un lenguaje de programación.
- Características de una descripción ABEL.
- Sentencias module para identificar un nombre de módulo, título y final.
- Sección de declaraciones:
 - Especificación de terminales de entrada y salida.
 - Especificación de nodos internos.
 - Sección de ecuaciones, incluyendo listado de la tabla de verdad.
- Descripciones ABEL operacional contra la de comportamiento.
- Especificación de condiciones irrelevantes en ABEL.
- Especificaciones jerárquicas y estructurales en ABEL.
- Arquitectura de ciertos PLD:
 - PAL16L8
 - PALCE16V8
 - CPLD XC9500 de Xilinx
- Arreglos de compuertas programables en campo.
- Empleo de especificaciones ABEL en el diseño.
- Herramientas de diseño asistido por computadora.
- Simulación de especificaciones ABEL.
- Síntesis en un formato lógico.
- Asignación del formato lógico para una especificación física.
- Simulación de la especificación física.
- Implementación ABEL por omisión.

PROBLEMAS

Conserve las descripciones ABEL que escriba para estos problemas; le serán de utilidad para efectuar los pasos adicionales que se piden al final.

1. Escriba una descripción ABEL de un restador de un solo bit.
2. Escriba una descripción ABEL de un sistema que compara dos números sin signo de 8 bits.

3. Escriba una descripción ABEL de un decodificador BCD a siete segmentos utilizando la especificación de tabla de verdad.
4. Sintetice las descripciones ABEL del problema 3 y la figura 11.
5. Verifique la descripción ABEL en la figura 11 comparando las dos implementaciones calculadas en el problema 4.
6. Dibuje el diagrama esquemático del sumador en serie en el ejemplo 1.
7. Sintetice la descripción ABEL del problema 2 utilizando varios valores diferentes para la orden @carry. Compile una tabla similar a la de la figura 30.
8. Escriba una descripción ABEL basada en transiciones de estado para la especificación dada en el problema 1 del capítulo 6, y sintetícela. Compare su resultado con la solución del capítulo 6.
9. Escriba una descripción ABEL basada en una tabla de transición para la especificación dada en el problema 2 del capítulo 6, y sintetícela. Compare su resultado con la solución del capítulo 6.
10. Escriba una descripción ABEL basada en ecuaciones del estado siguiente para la especificación dada en el problema 3 del capítulo 6 y sintetícela. Compare su resultado con la solución del capítulo 6.
11. Escriba una descripción ABEL basada en la transición de estado para la especificación dada en el problema 4 del capítulo 6, y sintetícela. Compare el resultado con su solución del capítulo 6.
12. Escriba una descripción ABEL basada en la tabla de transición para la especificación dada en el problema 5 del capítulo 6, y sintetícela. Compare su resultado con la solución del capítulo 6.
13. Escriba una descripción ABEL basada en las ecuaciones del estado siguiente para la especificación indicada en el problema 6 del capítulo 6, y sintetícela. Compare su resultado con la solución del capítulo 6.
14. De los tres métodos para especificar circuitos secuenciales en ABEL (transición de estados, tabla de estados y ecuaciones del estado siguiente), ¿cuáles son descripciones de comportamiento? Liste los métodos en orden de mayor o menor abstracción. El método más abstracto es el que especifica el menor detalle de una implementación. ¿Cuál método de especificación encuentra usted más eficiente: cuál requiere menos de su tiempo?
15. Escriba una descripción ABEL para la especificación que se indica en el problema 8 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
16. Escriba una descripción ABEL para la especificación que se indica en el problema 13 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
17. Escriba una descripción ABEL para la especificación que se indica en el problema 14 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
18. Escriba una descripción ABEL para la especificación que se indica en el problema 17 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
19. Escriba una descripción ABEL para la especificación que se indica en el problema 20 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
20. Escriba una descripción ABEL para la especificación que se indica en el problema 21 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
21. Escriba una descripción ABEL para la especificación que se indica en el problema 22 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
22. Escriba una descripción ABEL para la especificación que se indica en el problema 23 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
23. Escriba una descripción ABEL para la especificación que se indica en el problema 25 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
24. Escriba una descripción ABEL para la especificación que se indica en el problema 27 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
25. Escriba una descripción ABEL para la especificación que se indica en el problema 41 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.
26. Escriba una descripción ABEL para la especificación que se indica en el problema 44 del capítulo 6, y sintetícela. Compare su resultado con el de la solución del capítulo 6.

Organización de la computadora

Hasta ahora el alcance de este libro ha incluido ciertos componentes de los sistemas digitales: sumadores, multiplexores, flip-flops, registros, contadores, dispositivos lógicos programables, circuitos secuenciales (máquinas de estados) y similares. En este capítulo nos adentramos en lo que podría considerarse un cambio de escala: la organización de sistemas de una computadora digital completa. Los principios del diseño lógico digital y de la organización de los sistemas de computadora podrían parecer dos temas diferentes. Sin embargo, una computadora digital es simplemente una máquina de estados finitos, aunque muy complicada. Es por ello natural presentar la organización de la computadora digital como una extensión de los principios del diseño lógico digital que se describieron en los capítulos anteriores.

Recuerde del capítulo 8 que es posible dar dos descripciones de un sistema digital: una descripción de comportamiento y una estructural.

Una descripción *a nivel de transferencia de registros (RTL)* es una descripción de comportamiento que especifica algunas actividades de la máquina, ciclo de reloj por ciclo de reloj. Si usted piensa al respecto, quizá concluya que la descripción de la tabla de estados, que es una de comportamiento, constituye un ejemplo de una descripción RTL. En este capítulo trataremos con una descripción RTL de un sistema digital, aunque no entraremos en mucho detalle.

1 UNIDADES DE CONTROL Y DE TRAYECTORIA DE DATOS DE UN PROCESADOR

Una de las funciones que una computadora digital efectúa es el almacenamiento de información en memoria. Si usted tuviera que asignar un nombre a la unidad donde esta tarea se lleva a cabo, ¿no la llamaría “unidad de memoria” o “sección de memoria” o algún otro nombre de este tipo? En la unidad de memoria: también se almacenan las instrucciones para efectuar las operaciones deseadas. La función de la computadora es *procesar* tanto la información almacenada como cualquier otra que se suministra de manera externa, introducida mediante entradas primarias. Si tuviera que nombrar la parte de la máquina que efectúa este procesamiento, ¿acaso no la llamaría *procesador* o *unidad de procesamiento*? Cuando la computadora se encuentra en operación, los datos se intercambian en ambas direcciones entre el procesador y la memoria.¹

Para facilitar la descripción, resulta útil descomponer el procesador en dos bloques generales, denominados la *unidad de control* y la *unidad de flujo de datos* o *de trayectoria de datos*, como se muestra en la figura 1. Algunas de las entradas primarias al procesador se dirigen hacia la unidad de control y algunas a la unidad de trayectoria de datos. Esta última contiene circuite-

¹ Un componente fundamental de cualquier computadora digital es la *unidad de procesamiento central (CPU)*. En secciones posteriores se expondrá más de este tema.

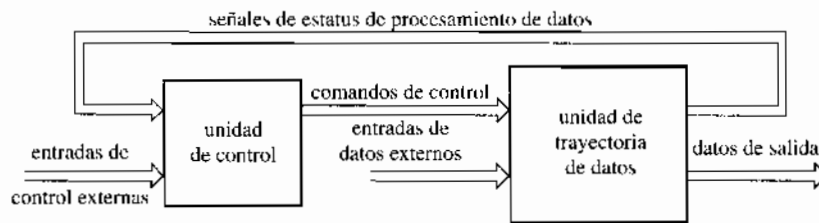


Figura 1. Estructura general de sistema digital.

ría que manipula los datos para calcular un resultado aritmético o lógico deseado. El nombre de esta unidad proviene de su organización como un conjunto de trayectorias a través de las cuales fluyen los datos. La unidad de control ordena a la unidad de trayectoria de datos efectuar operaciones específicas en los datos, y controla la secuencia de estas operaciones.

Además de las entradas primarias, la unidad de trayectoria de datos también recibe entradas de la unidad de control, como se observa en la figura 1. Las últimas son los comandos de control con los cuales se efectúa el procesamiento de datos. A su vez, además de las entradas externas (primarias) que controlan los detalles de las operaciones de procesamiento de datos, la unidad de control recibe entradas de la unidad de trayectoria de datos en lo referente al estatus de las operaciones que se están realizando ahí. Las únicas salidas de la máquina compuesta vienen de la unidad de trayectoria de datos.

Unidad de trayectoria de datos

Una unidad de trayectoria de datos incluye tanto circuitos combinatorios como secuenciales en una estructura ordinaria. Sus elementos combinatorios quizá incluyan decodificadores, multiplexores, sumadores y otros circuitos; entre sus elementos secuenciales se cuentan registros de todo tipo, pero no elementos de almacenamiento individuales (flip-flops). Cualquier componente del sistema que efectúa alguna función, como un sumador o una unidad lógica y aritmética (ALU), se conoce como *unidad funcional*.

La complejidad de la unidad de trayectoria de datos depende del número de registros, el número de unidades funcionales, sus interconexiones, y otros factores, que posiblemente variarán de manera sustancial de una unidad a otra. La trayectoria de datos es un circuito genérico capaz de ser configurado para efectuar cualquier número de operaciones. Contiene estructuras ordinarias, pero no la circuitería requerida para elegir una operación particular. Una secuencia de operaciones de trayectoria de datos conduce a la realización de algún resultado deseado.

Si bien todos los circuitos secuenciales pueden representarse como máquinas de estados finitos descritas mediante gráficas de estados o tablas de transición, aquellos que constituyen más que un puñado de estados no pueden representarse como tales de modo eficiente. Un buen ejemplo de un circuito de estas características es un registro. Un registro simple de 8 bits tiene 256 estados posibles. Debido a su estructura ordinaria, los circuitos lógicos combinatorios que efectúan las operaciones aritméticas (por ejemplo, sumadores multibit) no son candidatos para la minimización lógica. En otras palabras, ya conocemos buenas implementaciones de sumadores, como se explicó en el capítulo 4; no es necesario efectuar procedimientos de minimización lógica en sus especificaciones.

Es posible separar un procesador en dos partes: una consiste en unidades que tienen estructuras ordinarias, que se sintetizan con facilidad. La segunda parte cuenta con una estructura irregular y requiere algún esfuerzo para la síntesis.

Unidad de control

La unidad de control es una máquina de estados que ordena a la unidad de trayectoria de datos, seguir una serie de operaciones que culminan en un resultado deseado. Esta máquina de estados se sintetiza utilizando las técnicas del capítulo 6. Algunas veces se conoce como *lógica aleatoria* debido a que, a diferencia de la unidad de trayectoria de datos, no tiene una estructura ordinaria (no cuenta con registros, sólo elementos de almacenamiento individuales). También, a diferencia de la trayectoria de datos, la especificación de la lógica combinatoria de la unidad de control requiere minimización lógica; constituye un buen candidato para implementarse en lógica programable, como un PLA.

En el diseño de un procesador existen compromisos fundamentales entre la complejidad de la unidad de trayectoria de datos y la correspondiente a la unidad de control. Una unidad de trayectoria de datos simple requiere por lo común de control complejo. Por ejemplo, si sólo hay una conexión entre todos los registros de la trayectoria de datos, la unidad de control debe proporcionar una secuencia de señales de control para multiplexar los datos en la conexión. Por otro lado, una trayectoria de datos compleja requiere muchas veces de control simple debido a que los datos independientes pueden manipularse en paralelo. Así, no es posible que el proceso de diseño de las unidades de trayectoria de datos y de control sea independiente.

Ejemplo de multiplicador en serie

Completaremos la explicación anterior considerando un ejemplo bastante simple: el diseño de un circuito secuencial que calcula el producto de dos números de 4 bits. A continuación presentamos una ilustración numérica.

0101 (5)	multiplicando
$\times 1101$ (13)	multiplicador
0101	producto parcial 0
0000	producto parcial 1
0101	producto parcial 2
0101	producto parcial 3
01000001 (65)	producto

El sistema que efectúa esta operación es una unidad de procesamiento. Es posible diseñarla de diversas maneras; una de ellas efectúa una secuencia de operaciones de suma y corrimiento. El método de suma y corrimiento constituye una implementación directa de la secuencia de adición especificada por el arreglo precedente de productos parciales. Incluso en este simple caso, la implementación del multiplicador está separada en unidades de trayectoria de datos y de control. La trayectoria de datos contiene registros y un sumador; la unidad de control es una máquina de estados finitos que secuencía la unidad de trayectoria de datos a través de cada una de las operaciones de suma y corrimiento hasta que se acumulan todos los productos parciales.

Debido a que el control es dependiente de la arquitectura de la trayectoria de datos, el diseño del multiplicador debe empezar con la trayectoria de datos; primero, resulta necesario identificar:

- Los tipos de operación que se efectuará en la trayectoria de datos.
- Los registros necesarios.
- Las interconexiones (buses) entre unidades funcionales (por ejemplo, sumadores) y registros.

Utilizaremos dos registros de 4 bits y un registro de 8 bits en la trayectoria de datos para este diseño.

Es necesaria una descripción de la secuencia de operaciones efectuada por este circuito de multiplicación; una posibilidad es:

1. Asignar valores iniciales a todos los registros (producto $\leftarrow 0$, multiplicador \leftarrow entrada, multiplicando \leftarrow entrada).
2. Sumar el primer producto parcial al registro del producto.
3. Desplazar (correr) el registro del producto.
4. Sumar el segundo producto parcial al registro del producto.
5. Desplazar el registro del producto.
6. Sumar el tercer producto parcial al registro del producto.
7. Desplazar el registro del producto.
8. Sumar el cuarto producto parcial al registro del producto.
9. Desplazar el registro del producto.

La notación $A \leftarrow 0$ se encuentra en la notación de lenguaje de transferencia de registros (RTL); ésta especifica la transferencia del valor numérico cero al registro A. La larga descripción anterior de la operación del multiplicador puede escribirse de manera más concisa, como sigue.

1. Asignar los valores iniciales a todos los registros (producto $\leftarrow 0$, multiplicador \leftarrow entrada, multiplicando \leftarrow entrada).
2. Para cada bit j del multiplicador:
 - Sumar el producto parcial j -ésimo al registro del producto.
 - Desplazar el registro del producto.

Necesitan especificarse aún numerosos detalles de la implementación, pero esta descripción del hardware ayuda a visualizar la estructura de la trayectoria de datos y la máquina de estados necesaria para controlar su operación. En la figura 2 se muestra un diagrama de bloques de la trayectoria de datos; contiene dos registros de 4 bits, un registro de 8 bits, un sumador de 4 bits, una unidad funcional para generar un producto parcial y un flip-flop para almacenar el acarreo de salida del sumador.

Los estados de los registros en la trayectoria de datos relativos a unos cuantos pasos del algoritmo se muestran en la figura 3. Advierta que los valores irrelevantes se introducen en localidades donde el valor del bit de registro no es importante. Esto permite identificar redundancia en la trayectoria de datos. En particular, es posible utilizar los bits más a la derecha del registro del producto para almacenar el multiplicador, pues resulta viable descartar los bits examinados del multiplicador. Además, el contenido del registro del multiplicando nunca cambia. Si imponemos el requerimiento de que estas entradas se mantendrán constantes durante la operación, entonces este registro también puede eliminarse.

Ejercicio 1. Complete los pasos que faltan en la figura 3. ♦

Un nuevo diagrama de bloques que contiene únicamente un registro de corrimiento de 8 bits se muestra en la figura 4. En este caso se necesita otro componente (un multiplexor) de modo que a los 4 bits menos significativos del registro del producto puedan asignárseles valores iniciales con la entrada del multiplicador. El generador de productos parciales contiene cuatro compuertas AND; es factible recurrir a cualquier diseño de sumador (véase el capítulo 4) para la acumulación de los productos parciales.

Ejercicio 2. Determine los estados de los registros de trayectoria de datos en cada paso del procedimiento de multiplicación para un multiplicando de 0101 y un multiplicador de 1101. ♦

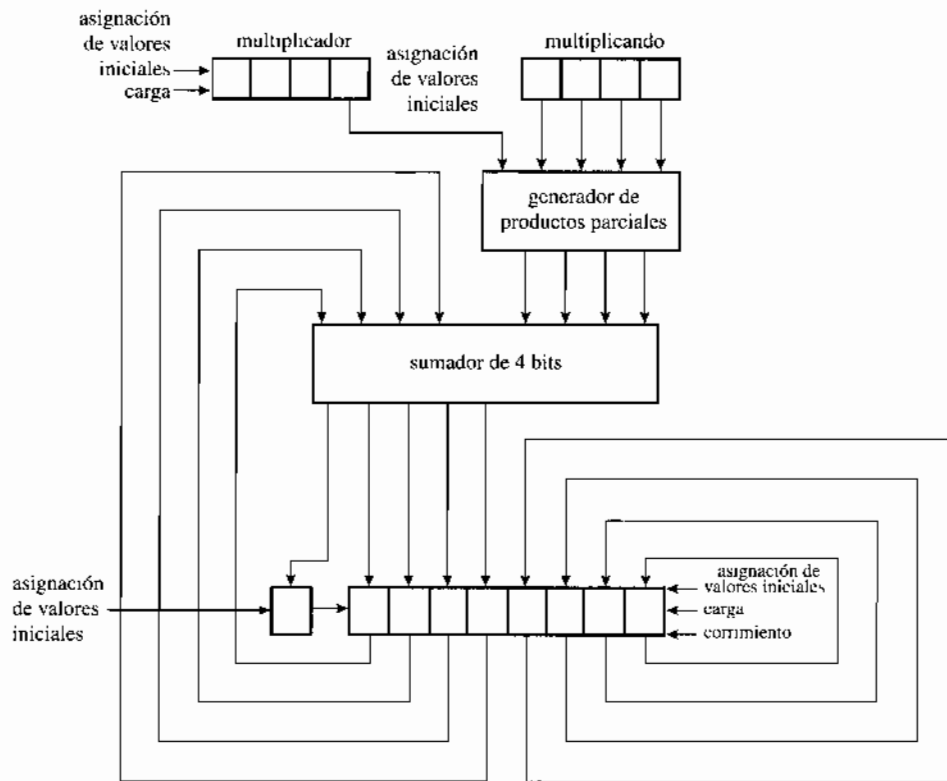


Figura 2. Trayectoria de datos de un multiplicador de suma y corrimiento.

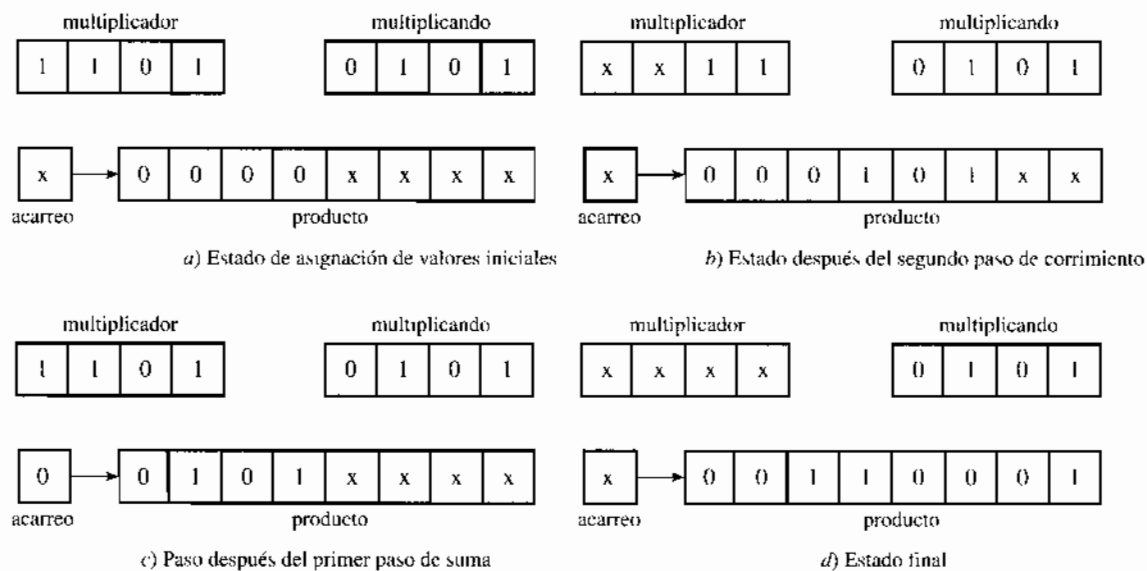


Figura 3. Estados de los registros de la trayectoria de datos del multiplicador de suma y corrimiento para un conjunto de entradas para pasos seleccionados del algoritmo.

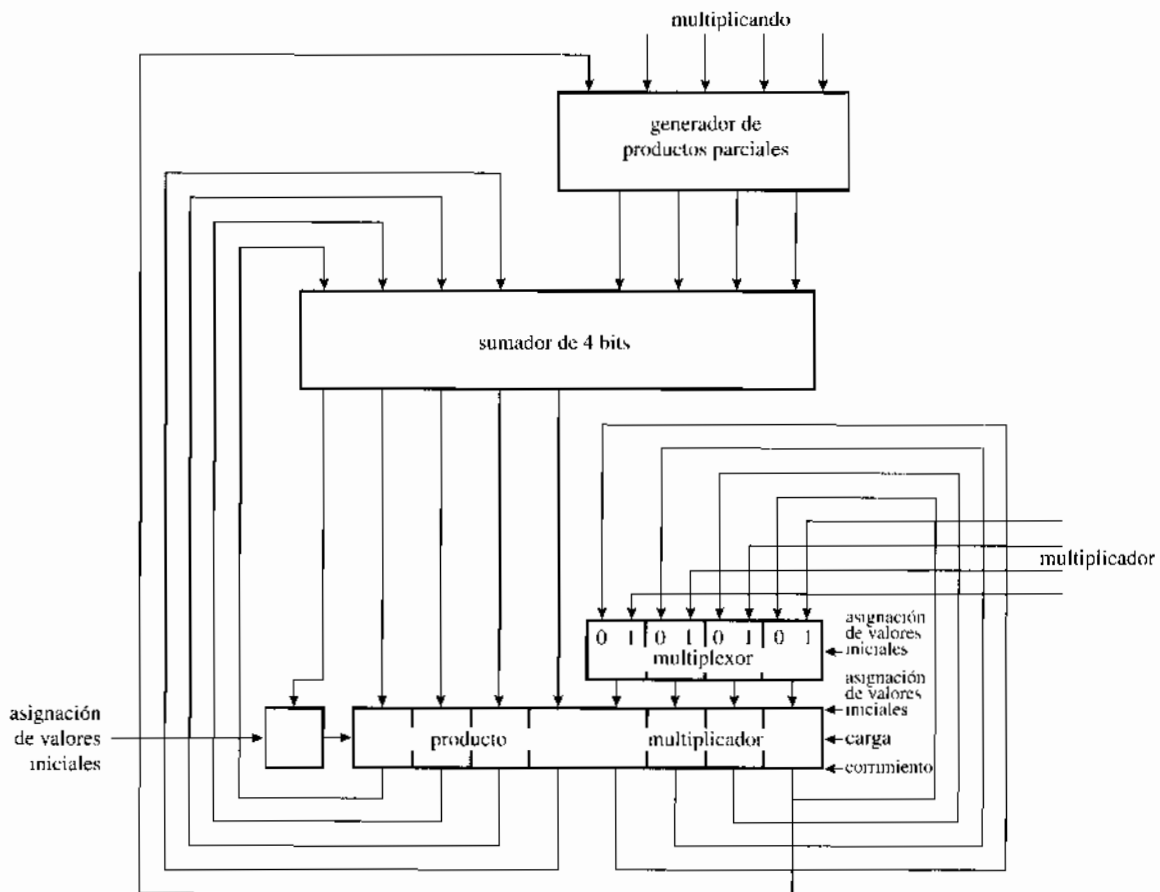


Figura 4. Arquitectura de trayectoria de datos alternativa para el multiplicador de suma y corrimiento.

En el momento apropiado durante la secuencia del algoritmo, la unidad de control para el multiplicador debe activar ciertas señales de control para la unidad de trayectoria de datos para cada una de las operaciones: asignación de valores iniciales, carga y corrimiento. El diagrama de estado del controlador se muestra en la figura 5; se trata de una implementación directa del control secuencial esbozado en el algoritmo precedente. El controlador tiene una sola entrada (la puesta a 0 o restablecimiento del hardware) y tres salidas.

Es posible elegir diferentes estrategias de implementación para la máquina de estados de la unidad de control; sin embargo, su estructura sugiere que el contador constituye una buena elección. ¿El número de estados es suficientemente pequeño de modo que pueda utilizarse un contador BCD? Reflexione antes de ver la respuesta.

Respuesta²

Elegiremos el 74LS90 para la implementación de este controlador; éste tiene el diagrama lógico y la descripción funcional que se indican en la figura 6. La asignación de estados debe elegirse para utilizar la funcionalidad incorporada del 74LS90. Así, a los estados A, B, C, ... se asignan las codificaciones binarias 0000, 0001, 0010, ..., respectivamente. La tabla de transición de estados del controlador y los requerimientos de excitación para el 74LS90 se dan en la figura 7.

² Hay nueve estados, por lo que resulta adecuado un contador BCD

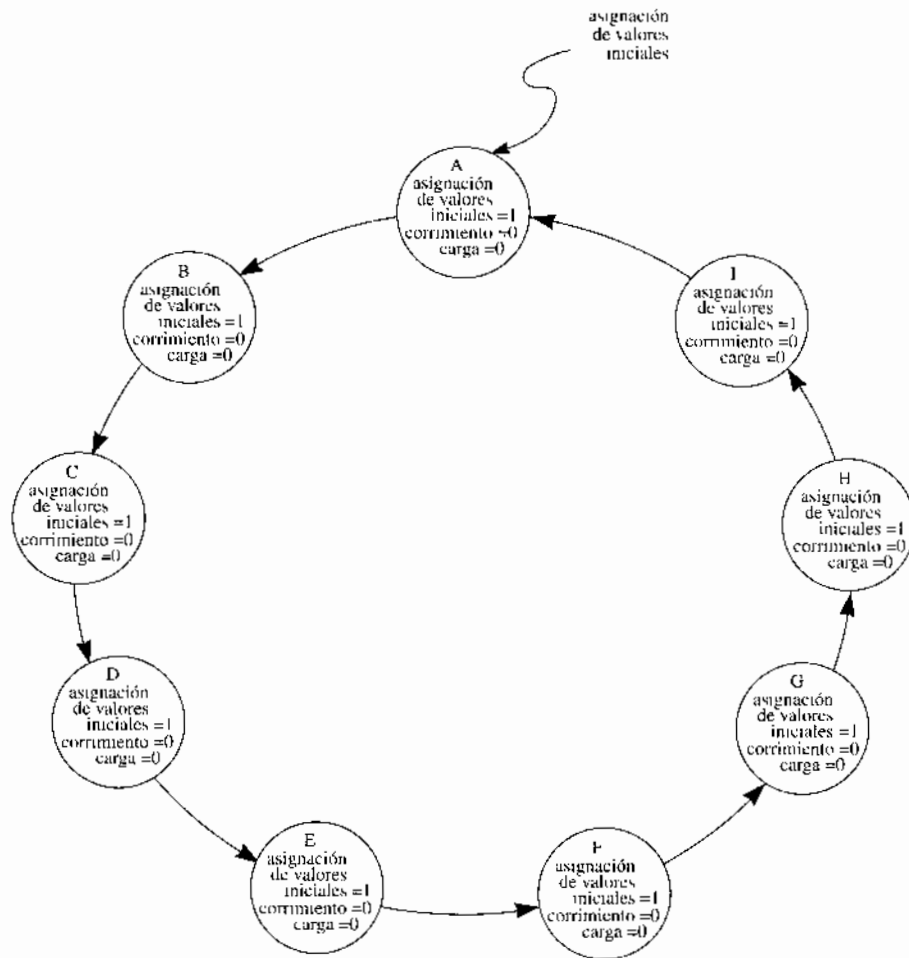


Figura 5. Diagrama de estados de la unidad de control para el multiplicador de suma y corrimiento.

¿Puede usted determinar si la máquina de estados finitos descrita en la figura 7 es del tipo de Mealy o de Moore? ¿Necesariamente tiene que ser una u otra, o es posible rediseñarse para que sea cualquiera? ¿Qué restricciones establece el uso del 74LS90 en el tipo (Mealy o Moore) de implementación?

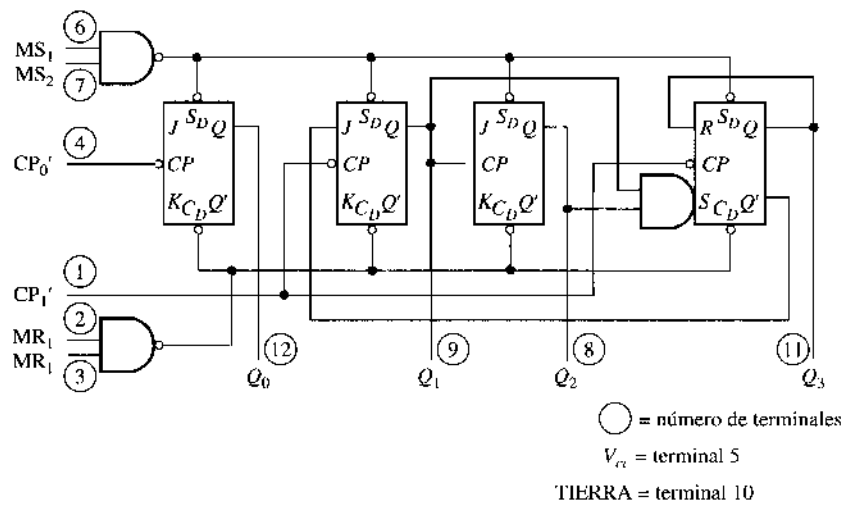
Respuesta³

El diagrama esquemático lógico de la implementación del multiplicador de suma y registro se muestra en la figura 8.

Ejercicio 3. Verifique la expresión lógica para las excitaciones del 74LS90 y las salidas de la unidad de control de la figura 7. ♦

Ejercicio 4. Utilizando los parámetros que se obtuvieron de las hojas de datos de los componentes en la figura 8, calcule el retardo de propagación del peor caso (crítico) en la trayectoria de datos. ¿Es importante el retardo de propagación crítico en la unidad de control? ♦

³ El examen cuidadoso de las especificaciones del 74LS90 (figura 6) revela que nuestra máquina es una Mealy y que no puede ser de otro tipo, ya que las entradas de restablecimiento son asíncronas. Esto es, el estado del 74LS90 puede cambiarse a 0000 sin un evento de reloj aplicando las entradas de restablecimiento adecuadas.



a)

Entradas de restablecimiento/establecimiento				Salidas				Conteo	Salida			
MR ₁	MR ₂	MS ₁	MS ₂	Q ₀	Q ₁	Q ₂	Q ₃		Q ₀	Q ₁	Q ₂	Q ₃
H	H	L	×	L	L	L	L	0	L	L	L	L
H	H	×	L	L	L	L	L	1	H	L	L	L
×	×	H	H	H	L	L	H	2	L	H	L	L
L	×	L	×					3	H	H	L	L
×	L	×	L					4	L	L	H	L
L	×	×	L					5	H	L	H	L
×	L	L	×					6	L	H	H	L
								7	H	H	H	L
								8	L	L	L	H
								9	H	L	L	H

H = nivel de voltaje alto
 L = nivel de voltaje bajo
 × = Valor irrelevante

b)

c)

Nota : La salida Q₀ se conecta a la entrada CP₁ para el conteo BCD.

Figura 6. Diagrama de bloques del 74LS90 y especificaciones funcionales. a) Diagrama de bloques. b) Selección de modo. c) Secuencia de conteo BCD.

	Estado presente					Entrada (restable- cimiento)	Estado siguiente				Asignación de valores iniciales	Salidas		Excitaciones			
	Q_3	Q_2	Q_1	Q_0	Q_3^+		Q_2^+	Q_1^+	Q_0^+	Carga		Corri- miento	MR_1	MR_2	MR_1	MR_2	
	×	×	×	×	1	0	0	0	0	×	×	×	1	1	0	×	
A	0	0	0	0	0	0	0	0	1	1	0	0	0	×	0	×	
B	0	0	0	1	0	0	0	1	0	0	1	0	0	×	0	×	
C	0	0	1	0	0	0	0	1	1	0	0	1	0	×	0	×	
D	0	0	1	1	0	0	1	0	0	0	1	0	0	×	0	×	
E	0	1	0	0	0	0	1	0	1	0	0	1	0	×	0	×	
F	0	1	0	1	0	0	1	1	0	0	1	0	0	×	0	×	
G	0	1	1	0	0	0	1	1	1	0	0	1	0	×	0	×	
H	0	1	1	1	0	1	0	0	0	0	1	0	0	×	0	×	
I	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	×	

Figura 7. Tabla de transición de estados de la unidad de control y requerimientos de excitación del 74LS90.

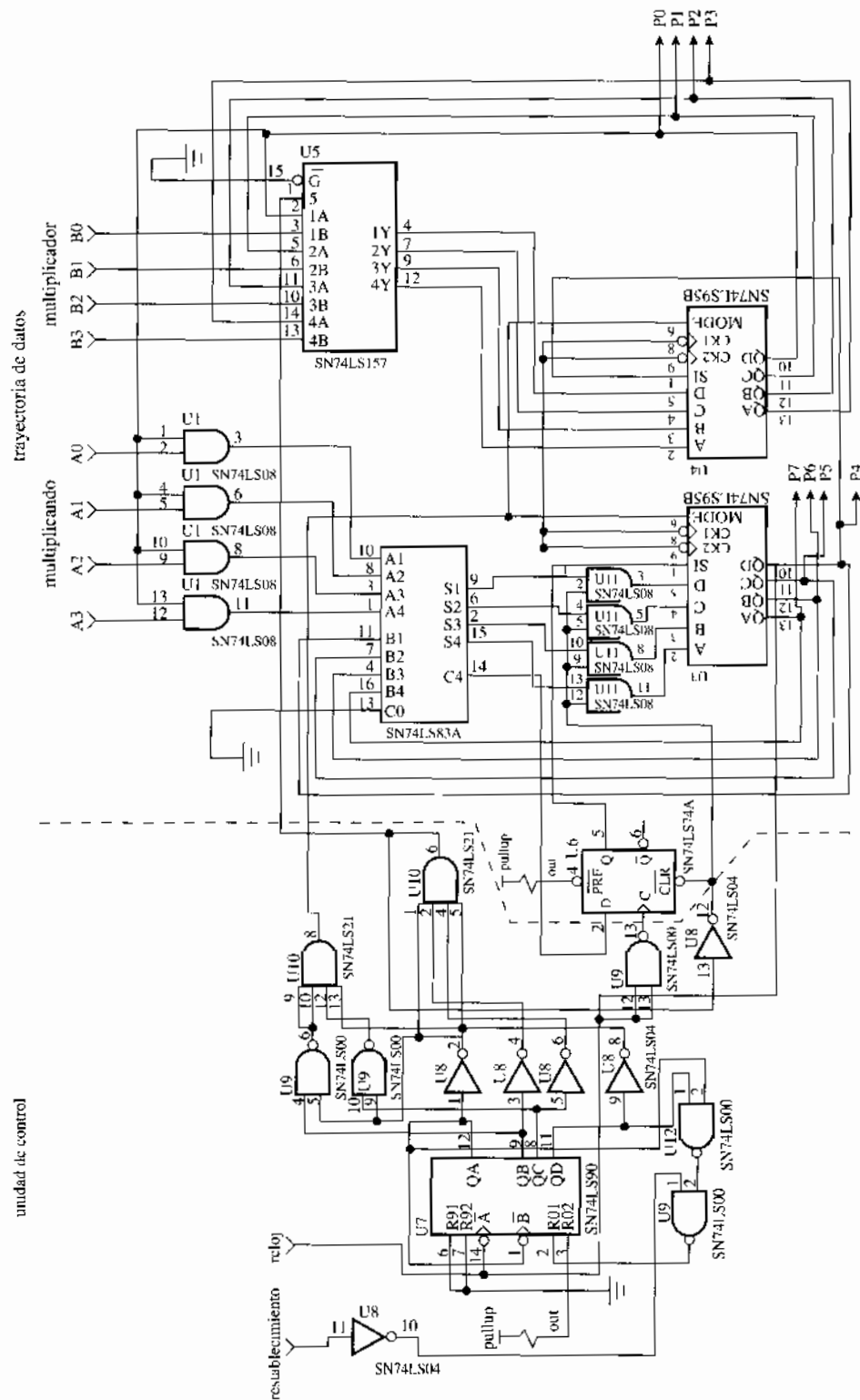


Figura 8. Diagrama esquemático del multiplicador completo de suma y corrimiento.

2 COMPUTADORA BÁSICA DE PROGRAMA ALMACENADO

El sistema digital más común sobre la tierra, cuando este libro aparece, a principios del siglo veintiuno, es la computadora digital.

No es posible que nadie que lea este libro sea ajeno a la ubicua computadora personal. La actual computadora digital de programa almacenado fue concebida por John von Neumann, quien propuso que la máquina se organizara de manera tal que pudiera implementarse un procedimiento mediante la ejecución de una secuencia de instrucciones almacenadas en una ubicación central (memoria).⁴

Las instrucciones de las tareas que efectuará la computadora se almacenan en la *memoria*. En la *unidad de procesamiento central* (CPU) es donde se efectúan (*ejecutan*) las tareas en varios pasos. Una secuencia de pasos se realiza repetidamente:

- La máquina *busca* en secuencia las instrucciones de la memoria en el CPU.
- Las instrucciones se *decodifican* después en el CPU.
- Por último, las instrucciones se efectúan, o *ejecutan*, de nuevo en el CPU.

Este simple proceso de tres pasos se conoce como el ciclo de *búsqueda/decodificación/ejecución*. Mientras la computadora está operando, este proceso de tres pasos se efectúa de manera repetida a una velocidad de millones de veces por segundo.⁵ Es lo primero que una computadora hace cuando se pone en operación, y nunca deja de hacerlo. Puesto que éste es un proceso fijo, puede implementarse en el hardware (es decir, como la unidad de control de una unidad de procesamiento).

Una computadora efectúa diferentes tareas especificando las instrucciones apropiadas y su secuencia. Como esta parte del control de la computadora es programable, se especifica en el software. La computadora debe evidentemente tener algunos componentes de hardware para habilitar la ejecución de las instrucciones de software: la trayectoria de datos. La única instrucción que debe cablearse en la computadora es la que permita localización de la instrucción del principio. Las computadoras personales contemporáneas cuentan con hardware que fuerzan a la máquina a tener acceso a la primera instrucción desde un sistema operativo integrado (BIOS).

El BIOS contiene instrucciones que inician el procedimiento de arranque de la computadora. El programa BIOS ordena primero a la computadora la verificación, en la unidad manejadora de discos flexibles, de la presencia de un disco que contenga el archivo de arranque del sistema operativo. Si no hay disco en la unidad, el BIOS empieza a buscar en el disco duro del sistema este archivo. Si se encuentra un disco flexible en la unidad y éste no contiene un archivo de arranque del sistema operativo, la máquina envía un mensaje de error y pide un disco de arranque válido. Si encuentra un archivo de arranque del sistema operativo en el disco flexible en la unidad, entonces arranca la computadora con ese sistema operativo. Cada una de las instrucciones que se ejecutan durante este proceso se lleva a cabo por medio del ciclo búsqueda de instrucción, decodificación y ejecución.

⁴ El húngaro John von Neumann (1903-1957) recibió el doctorado en ingeniería química a los 23 años de edad, de aunque enseñó matemáticas en la Universidad de Berlín antes de ingresar a Princeton, a su Instituto para Estudios Avanzados. Realizó aportaciones importantes en muchas áreas, pero no en ingeniería química. Su trabajo en la MANIAC-I en Princeton estableció los cimientos para el diseño de todas las computadoras programables subsecuentes.

⁵ Se espera que para el año 2005, la velocidad de ejecución alcance miles de millones de instrucciones por segundo.

Unidad de procesamiento central (CPU)

El corazón del sistema de computadora es la unidad de procesamiento central (CPU).⁶ El microprocesador es sólo un tipo de CPU; ejemplos de otros tipos son los *microcontroladores* y los *procesadores de señales digitales*. Un sistema digital puede clasificarse como un CPU si contiene la circuitería necesaria para ejecutar un conjunto de instrucciones definidas que habilitan cualquier computación determinística deseada. ¿Pero cuál es el número mínimo de instrucciones necesarias? ¿Es posible que un CPU simple se especifique de modo que pueda construirse en el laboratorio en unas cuantas horas o que sea factible adaptarlo en un PLD de la serie 9500 de Xilinx? Investigaremos estas interrogantes.

Trayectoria de datos simple

Como se explicó en la sección 1, la circuitería de un CPU se separa en una unidad de trayectoria de datos y en una unidad de control.

Consideraremos primero la trayectoria de datos de un CPU y planteamos la pregunta, ¿qué cantidad mínima de hardware se necesita para constituir una trayectoria de datos viable? Por definición, un CPU debe leer una instrucción localizada en la memoria; esto se conoce como una *búsqueda de instrucción*. Para que el CPU realice lo anterior, debe haber al menos algún medio de generar y almacenar la dirección de memoria a partir de la cual buscar la instrucción. Un registro es adecuado para cuestiones de almacenamiento; necesitamos uno que pueda *cargarse* con lo que se va a almacenar y que pueda *incrementarse* para ir a la siguiente tarea.

Las instrucciones en la computadora se ejecutan por lo común en secuencia. Por consiguiente, cuando se va a buscar la siguiente instrucción, es posible incrementar el registro. No en todos los casos es en realidad necesario ejecutar la siguiente instrucción más próxima en una secuencia. Considere *lazos* en un programa, esto es, secuencias de tareas que regresan al punto de inicio y que luego se repiten. Para volver al principio de un lazo, no se incrementará simplemente la dirección de la instrucción; necesitamos medios para cargar una dirección arbitraria en el registro para alterar la secuencia de ejecución consecutiva. Este registro a menudo recibe el nombre de *contador de programa* (PC) debido a que

- Es un contador.
- Apunta a la localidad de memoria donde se ejecuta el programa.

Una instrucción que carga al contador de programa con la dirección de la siguiente instrucción se conoce como instrucción de *bifurcación* (o instrucción de *salto*) debido a que la ejecución del programa se bifurca hacia una instrucción que es diferente de la siguiente instrucción en secuencia. No puede esperarse que el CPU busque una instrucción, la decodifique y luego la ejecute en la totalidad de un ciclo de reloj. Entonces, requerimos de un registro para el almacenamiento de la instrucción, cuando el reloj envíe un pulso activación. Lo denominaremos el *registro de instrucción* (IR). Además, resultan necesarios

- Una unidad lógica y aritmética (ALU) para manipular datos.
- Un registro para almacenar datos.

Los datos que se almacenarán pueden provenir de la memoria o generarse dentro de la trayectoria de datos. Es posible almacenar los dos tipos de datos en registros diferentes; también es factible almacenarlos en los mismos registros. Para simplificar, así lo consideraremos aquí. Debido a que un registro de datos se usa para acumular resultados aritméticos, a menudo se le denomina *acumulador* (AC).

⁶ Un CPU de una computadora personal moderna compatible con IBM es el microprocesador Pentium, de manera más reciente el Pentium III. Para la computadora Macintosh de Apple el CPU es el microprocesador PowerPC.

Una dirección de memoria puede ser la dirección de una instrucción o la de un dato. Así, se necesita un registro, llamado *registro de dirección de memoria* (MAR), para servir como interfaz entre las fuentes de dirección múltiple y el bus que transporta las direcciones y se conecta a la memoria.

¿Qué es lo que se espera que sean las fuentes de direcciones de memoria en la trayectoria de datos? Evidentemente, el contador de programa es una fuente para las direcciones de instrucciones. La trayectoria de datos necesita también datos, por ejemplo, para una operación de suma. Las direcciones de datos están contenidas con frecuencia en la propia instrucción (ampliaremos este punto más adelante). Por ello es que en nuestro sistema el MAR puede tomar una dirección ya sea del contador de programa (para buscar una instrucción) o del IR (para buscar datos).

Éste es un esbozo de la cantidad mínima de hardware necesaria para ejecutar instrucciones que son útiles en la implementación de un programa; con eso, es posible delinear la arquitectura de la trayectoria de datos. También es necesario determinar el ancho de los datos y las direcciones de memoria en el sistema, así como los tipos de instrucciones que ejecutará el CPU. Por lo general, el costo y el tiempo restringen la complejidad. En este ejemplo, a fin de minimizar la complejidad, y subrayar los conceptos de la especificación y el diseño del CPU, elegimos anchos de datos y direcciones pequeñas.

Si bien tres es el número mínimo de tipos de instrucciones que se requiere para un CPU funcionalmente completo, elegimos cuatro instrucciones para este ejemplo. Las tareas que se efectuarán son cargar los datos a partir de la memoria (*instrucción de carga*), almacenar los datos en la memoria (*instrucción de almacenamiento*), alterar la secuencia de ejecución consecutiva (*instrucción de salto*) y efectuar una operación lógica entre dos operandos. (Elegiremos la operación lógica NAND, de modo que ésta se denominaría la instrucción *NAND*. ¿Por qué elegir NAND en lugar de, digamos, AND?)⁷

Para codificar cuatro instrucciones con datos binarios necesitamos 2 dígitos binarios. Una instrucción, por tanto, contendrá 2 bits de información que especifican la operación que llevará a cabo el CPU. Estos 2 bits se conocen comúnmente como el *código de operación*, o *código-op*, de la instrucción. Al elegir la longitud del código de operación, hemos empezado a definir el *formato de instrucciones* del CPU. Este formato especifica cómo escribir cada instrucción (carga, almacenamiento, etc.) en formato binario para el almacenamiento en la memoria de la computadora. En otras palabras, define con precisión el lenguaje del CPU o máquina. Éste es el origen del término *lenguaje de máquina*, que se refiere a las instrucciones binarias ejecutadas por un CPU.

¿Qué otro tipo de información se necesita en una instrucción en lenguaje de máquina? Si queremos tener acceso a los datos, necesitamos entonces un medio para direccionarlos. Si las instrucciones mismas contienen la dirección del dato, tiene sentido denominar este método *direccionamiento directo*.⁸ La totalidad de nuestras instrucciones requieren una dirección de memoria.

La instrucción de salto requiere una dirección que especifique la localidad en la memoria de la cual provendrá la siguiente instrucción; puede estar condicionado de manera que el salto ocurra únicamente bajo ciertas circunstancias específicas. En nuestra máquina saltaremos si el contenido del acumulador es cero.

La instrucción NAND debe contener la dirección de uno de los operandos. supondremos que el segundo operando es el acumulador. De este modo el formato de la instrucción NAND resulta consistente con las demás instrucciones. ¿Dónde estaría el lugar más apropiado para almacenar el resultado de la operación NAND, en la memoria o en el acumulador? Algo que debemos evitar es la necesidad de especificar una segunda dirección en la instrucción. Una necesidad de este tipo complicaría la organización de nuestra computadora; requeriría instrucciones que ocuparían

⁷ Debido a que la NAND es lógicamente completa, la AND no lo es (refiérase al capítulo 2).

⁸ Existen numerosos métodos para direccionar datos en la memoria. Es posible encontrarlos en libros acerca de arquitectura de computadoras, pues están más allá del objetivo de este libro.

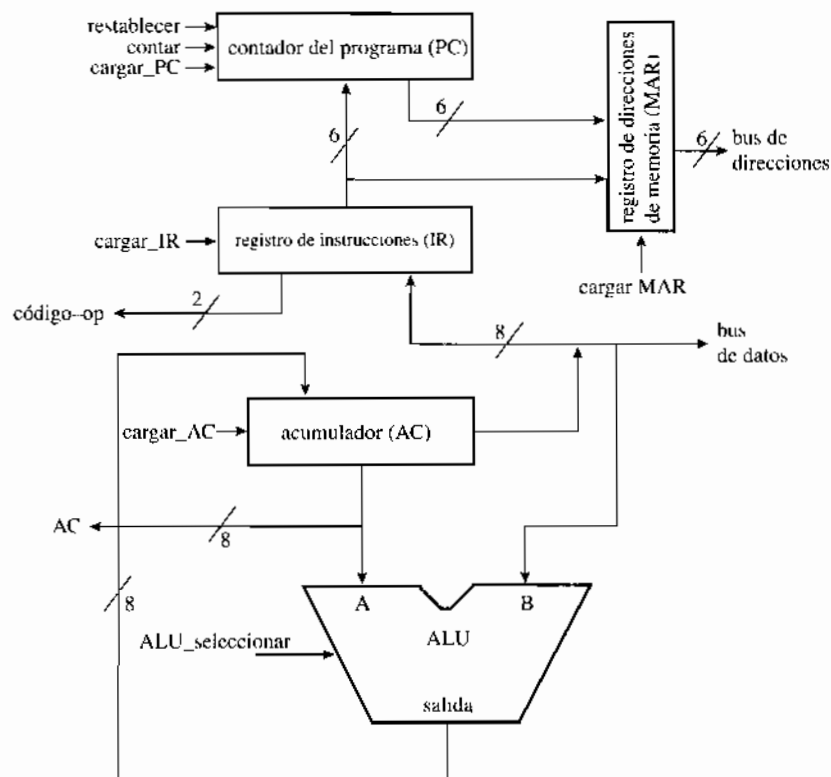


Figura 9. Arquitectura de la trayectoria de datos de un CPU simple.

dos palabras en la memoria, y además necesitaría más unidades complicadas de trayectoria de datos y control. En consecuencia, almacenaremos el resultado de la operación NAND en el acumulador.

Sería de utilidad si las instrucciones y los datos tuvieran la misma longitud. Suponiendo datos de 8 bits (y consecuentemente instrucciones de 8 bits), ¿cuál sería el ancho del bus de direcciones?

Respuesta⁹

La arquitectura de nuestra trayectoria de datos se muestra en la figura 9. Debe verse como una arquitectura preliminar, puesto que durante el diseño de la unidad de control quizá decidamos cambiar algunos detalles. Es claro que al considerar tanto las unidades de trayectoria de datos como de control, tienen que establecerse compromisos entre la velocidad y la complejidad.

Control de la trayectoria de datos simple

Las instrucciones para la máquina se ilustran en la figura 10. Cada una de éstas requiera quizá varios ciclos de reloj para su ejecución. Las operaciones efectuadas en un ciclo de reloj reciben el nombre de *microoperaciones*. La unidad de control se especifica identificando las microoperaciones que se efectuarán en cada estado de control. Por ejemplo, ciertas microoperaciones resultan necesarias para efectuar una búsqueda de instrucciones. Una de éstas es para informar a la memoria lo que queremos leer. Otra asegura que el dato enviado por la memoria es almacenado.

⁹ 6 bits. En este caso, todas las instrucciones caben en una localidad de memoria.

Instrucción	Descripción	Código-Op
Carga AC, memoria	Cargar AC con contenido de la localidad de memoria	00
Almacenar memoria, AC	Almacenar AC en la localidad de memoria	01
NAND AC, memoria	NAND de AC y del contenido de la localidad de la memoria	10
Saltar memoria	Saltar a la localidad de memoria si AC es 0	11

Figura 10. Instrucciones de máquina para un CPU simple.

La unidad de control debe activar la carga del registro de instrucciones Load_IR y emitir una petición de lectura de memoria para conseguir esto. (Recuerde que en lógica positiva, “activar” significa poner un 1 lógico.) Como resultado, ese dato es recibido mediante la trayectoria de datos.

A fin de confirmar que se está direccionando la localidad de memoria correcta en cualquier operación, la secuencia de control debe asegurar que, en alguna microoperación previa, la dirección correcta se puso en el MAR. Escribamos la secuencia de estados y microoperaciones que se efectuarán en cada estado para buscar, decodificar y ejecutar las instrucciones de la máquina. Suponga que un restablecimiento 0 del hardware lleva a la máquina al estado A. El estado A debe poner en 0 el contador de programa para señalar hacia la primera instrucción en la memoria. Como se explicó antes, esto puede escribirse en el RTL como $PC \leftarrow 0$. En el estado B de nuestra unidad de control el contenido del contador de programa debe transferirse al MAR de manera que la dirección pueda utilizarse para leer una instrucción en la memoria. La notación RTL para esta microoperación es $MAR \leftarrow PC$. Al mismo tiempo, dependiendo de la arquitectura de la trayectoria de datos, tal vez sea posible incrementar el contador de programa para que apunte a la siguiente instrucción. Hasta ahora la máquina de estado de nuestra unidad de control se encuentra así:

Estado A: $PC \leftarrow 0$, va al estado B

Estado B: $MAR \leftarrow PC$, $PC \leftarrow PC + 1$, va al estado C.

La última línea conduce al estado C; imaginemos que se hará en ese estado. La dirección de la instrucción se encuentra en el MAR, de modo que estamos listos para leerla. La memoria necesitará una señal de control que le indique que estamos leyendo datos de la dirección en el bus de direcciones en oposición a escribir datos hacia la dirección. Esta señal de control puede denominarse R/W' y activarse en el nivel alto ($R/W' = 1$) en una operación de lectura y en el nivel bajo ($R/W' = 0$) en la operación de escritura. Escriba las operaciones de transferencia de registro que deben efectuarse inmediatamente después.

Respuesta¹⁰

Cuando alcanzamos el estado D, la instrucción está en el IR, por lo que necesitamos examinar (decodificar) los bits del código de operación y ejecutar una secuencia de microoperaciones basadas en su valor. Por ejemplo, si el código de operaciones es 00, entonces se carga la instrucción, y requerimos poner la dirección del dato en el MAR y efectuar otra lectura de memoria. La secuencia de microoperaciones para carga y salto son como se muestra. Escriba usted las microoperaciones para las instrucciones restantes.

¹⁰ Estado C: $R/W' \leftarrow 1$, $IR \leftarrow$ bus de datos, ir al estado D.

- Estado D: Si código de operación = 00, entonces ir al estado E.
 Si código de operación = 01, entonces ir al estado F.
 Si código de operación = 10, entonces ir al estado G.
 Si código de operación = 11, entonces ir al estado H.
- Estado E: $MAR \leftarrow IR$, ir al estado Ea.
- Estado Ea: $R/W' \leftarrow 1$, $ALU_B \leftarrow$ Bus de datos, $ALU_seleccionar \leftarrow$ paso,
 $AC \leftarrow ALU_salida$, ir al estado B.
- Estado H: Si $AC = 0$, entonces $PC \leftarrow IR<5:0>$, ir al estado B.

Hay pocas cosas nuevas en la secuencia de estados anterior. El bus de datos no es una entrada directa al AC, por lo que es necesario pasar los datos a través de la ALU y dentro del AC. De ese modo, la ALU debe tener una operación que permita que los datos en su entrada B pasen directamente hacia la salida sin modificación. Cuando se completa el estado Ea, finaliza la ejecución de la instrucción, por lo que la unidad de control secuencia de regreso al estado B para preparar la búsqueda de la siguiente instrucción. Por último, en la instrucción de salto, los 6 bits menos significativos del IR se transfieren al contador de programa si el contenido del acumulador es cero. Usamos la notación $IR<5:0>$ para identificar los bits del 5 al 0 del registro de instrucciones.

Ejercicio 5. Escriba la notación RTL de las microoperaciones para las operaciones de almacenamiento y NAND de este CPU simple. (A continuación se presenta la respuesta, pero obténgala usted antes de consultarla.)

Respuesta: Estado F: $MAR \leftarrow IR$, ir al estado Fa.
 Estado Fa: $R/W' \leftarrow 0$, bus de datos $\leftarrow AC$, ir al estado B.
 Estado G: $MAR \leftarrow IR$, ir al estado Ga.
 Estado Ga: $R/W' \leftarrow 1$, $ALU_B \leftarrow$ Bus de datos, $ALU_elegir \leftarrow$ NAND,
 $AC \leftarrow ALU_salida$, ir al estado B. ♦

Constituye una buena práctica dividir la memoria del sistema en dos partes: una para las instrucciones y otra para los datos. El resultado se conoce como la *arquitectura Harvard*. La alternativa es la *arquitectura Princeton*, donde los datos y las instrucciones pueden coexistir en la misma parte de la memoria. La separación de las instrucciones y los datos en la memoria produce un nivel mínimo de protección para las instrucciones. Un programa nunca debe escribirse de modo que contenga instrucciones diseñadas para sobrescribirse en sí mismas. Si los datos y las instrucciones se mezclan en la memoria, es más probable que las instrucciones se sobrescriban de manera no intencional.

3 IMPLEMENTACIONES DE LA UNIDAD DE CONTROL

La unidad de control del CPU secuencia la trayectoria de datos a través de las microoperaciones; es una máquina de estados finitos, donde cada estado representa una microoperación. Puede implementarse utilizando cualesquiera de las técnicas presentadas en los capítulos anteriores (por ejemplo, compuertas, PLA, ROM). Existen dos implementaciones comunes, conocidas como *cableada* y *microprogramada*.

El control cableado se refiere a la implementación con compuertas y flip-flops. Se denomina así debido a que luego de que se construye, sólo puede modificarse cambiando el hardware.

El control microprogramado, por otro lado, se refiere a la implementación que utiliza una ROM y un secuenciador de microprograma.

- La ROM se utiliza para almacenar las señales de control en diversas localidades.
- El secuenciador de microprograma elige la localidad de la microoperación deseada dentro de la ROM.

Implementaremos la unidad de control de nuestro CPU simple utilizando ambas técnicas.

Unidad de control cableada

Es posible describir la unidad de control cableada utilizando cualquiera de las herramientas de especificación estudiadas antes en el libro: diagrama de estados, diagrama ASM o ABEL. Puesto que el uso del lenguaje de descripción de hardware es el método más común en la actualidad, escribiremos la unidad de control cableada en ABEL. Para cada estado esbozado en la sección anterior necesitamos activar las señales de control apropiadas que se indican en la figura 9. Cuando el PC está en el estado A, la señal de restablecimiento tiene que activarse y el estado siguiente es B. El código ABEL que especifica este comportamiento se muestra en la figura 11.

¿Cómo codificamos el estado B? No es problema activar la señal de control contar_PC, ¿pero cómo controlamos lo que se maneja con el reloj en el MAR? Hay dos entradas al MAR: IR y PC. Un registro con dos entradas generalmente no existe, pero podemos poner un multiplexor a la entrada y elegir PC o IR como la fuente.

Evidentemente, necesitamos refinar un poco la arquitectura de la figura 9. Esta tarea quizá requiera hardware adicional, tal como el multiplexor a la entrada del MAR, o tal vez necesite la adición de pasos de control extra para evitar conflicto en una conexión.

Ejercicio 6. Identifique cualquier ambigüedad o conflicto en la arquitectura de la figura 9. Dibuje un diagrama de bloques de una nueva arquitectura. (A continuación se presenta la respuesta; obtenga la propia antes de consultar la que se indica.)

Respuesta. El AC tiene dos destinos: la unidad de control y el bus de datos. Este último es bus bidireccional, por lo que al enviar un dato a la trayectoria de datos, la salida del AC no debe ocupar el bus. De tal modo, el AC debe tener salidas de tres estados, lo cual requiere una señal de control habilitadora de salida. (Véase el capítulo 2, sección 10.) Además, esto quiere decir que los datos no se pueden enviar en el bus de datos hacia el CPU mientras la salida del AC esté siendo leída por la unidad de control o la ALU. La unidad de control está interesada sólo en el

Module CPU_control

Title 'Unidad de control cableado'

"Unidad de control cableado de un CPU simple"

Declarations

Reloj restablecimiento código_up1..código_up0, AC7..AC0 **PIN**;

restablecer_PC, contar_PC, cargar_PC, cargar_IR, cargar_AC **PIN** **istype** 'com';

ALU_seleccionar, cargar_MAR, IR_PC, R_W **PIN** **istype** 'com';

Q3..Q0 **NODE** **ISTYPE** 'reg'; "salida con registro

A = [0, 0, 0, 0]; "asignación de estados

B = [0, 0, 0, 1];

C = [0, 0, 1, 0];

D = [0, 0, 1, 1];

Equations

[Q3..Q0].clk = reloj

[Q3..Q0].ar = restablecer; "restablecimiento asíncrono

state_diagram [Q3, Q2, Q1, Q0]

state A: restablecer_PC = 1; contar_PC = 0; cargar_PC = 0;

goto B; "No importa que sucede con los otros registros

end CPU_control

Figura 11. Inicio de la especificación ABEL de la unidad de control cableada de un CPU simple.

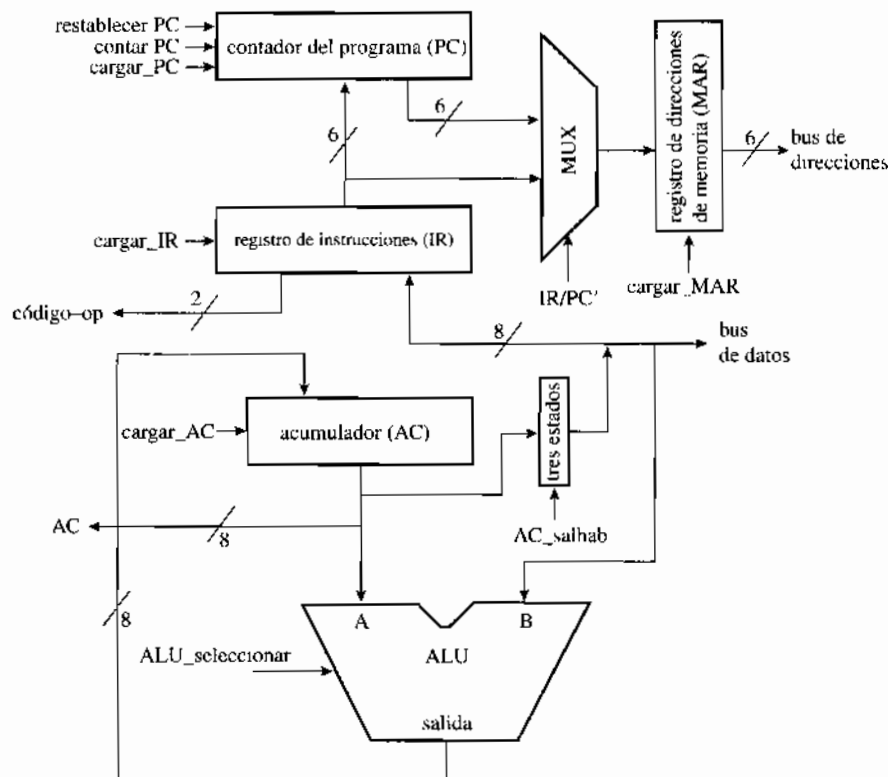


Figura 12. Arquitectura refinada de la figura 9.

contenido del AC en el estado H; el bus de datos no se utiliza en este estado. Sin embargo, en los estados Ea y Ga los datos se envían a la entrada ALU_B en el bus de datos, y desde el AC hasta la entrada ALU_A. Una forma de evitar el conflicto en el bus de datos consiste en insertar un búfer de tres estados entre la salida del AC y el bus de datos. (El conflicto al que nos referimos aquí a menudo se denomina *contención de bus* o *disputa de bus* debido a que dos fuentes de datos contienden por el uso del bus.) Cuando el AC va a pasar al bus, debe habilitarse el búfer de tres estados. En la figura 12 se muestra una arquitectura de trayectoria de datos modificada. Quizá existan varias formas más de modificar la arquitectura de la figura 9 para eliminar los conflictos. ♦

Continuamos ahora con la descripción ABEL de la unidad de control. En la figura 12 se presenta un diagrama de bloques refinado y detallado de la unidad de trayectoria de datos. Refiérase a él para todos los nombres de las señales de control que se utilizan en la siguiente descripción ABEL. El código para el estado B debe realizar dos cosas:

- Transferir el contenido del PC al MAR.
- Incrementar el PC

Surge una pregunta: ¿estas acciones pueden efectuarse de manera simultánea? Nuestro CPU es síncrono, por lo que suponemos que cada elemento de memoria se maneja por medio de reloj con la misma señal. Así, las señales de control establecen el estado presente del PC (el estado siguiente del MAR) a la entrada del MAR, y el estado siguiente del PC ($PC + 1$) a la entrada del PC. El evento de reloj transfiere el estado siguiente hacia el estado presente, los retardos de pro-

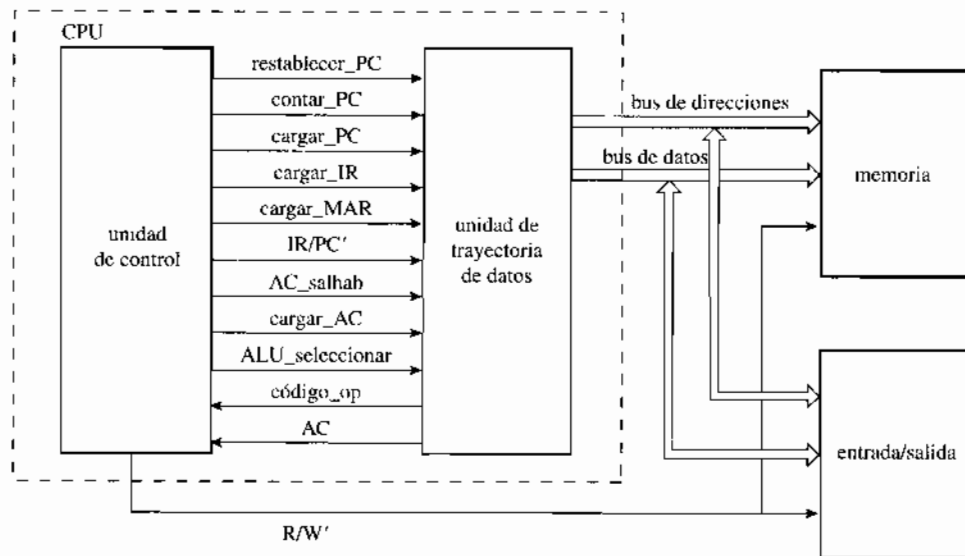


Figura 13. Diagrama de bloques a nivel de sistemas de una computadora simple que ilustra la memoria y la interfaz de E/S.

pagación satisfacen los tiempos de retención de los registros, y los nuevos estados se capturan en los registros de manera simultánea sin ningún problema. El código ABEL para el estado B se puede escribir como sigue:

Estado B: $\text{restablecer_PC} = 0$, $\text{cargar_PC} = 0$, $\text{contar_PC} = 1$, $\text{IR/PC}' = 0$,
 $\text{cargar_MAR} = 0$, $\text{cargar_AC} = 0$, ir al estado C.

¿Por qué especificamos que nada se cargará en el AC en el estado B?

Memoria e interfaz de E/S

El código ABEL para el estado C requiere la activación de la señal de control R/W' , la cual no ha aparecido hasta ahora en alguno de nuestros diagramas de bloque. Ésta es una señal enviada directamente desde la unidad de control hasta la memoria principal del sistema.

Puesto que la memoria principal no se considera parte del CPU, no se muestra en la figura 12. Un diagrama de bloques de alto nivel del sistema se muestra en la figura 13. En esta figura, los dispositivos de entrada/salida (E/S) se conectan al CPU al igual que a la memoria.

Se afirma que un sistema configurado de esta manera tiene *E/S mapeados a memoria*. Las instrucciones que se utilizan para leer y escribir en los dispositivos de E/S son los mismos que los utilizados para la memoria; éstos se direccionan de la misma forma que las localidades de memoria individuales.

Algunos CPU comerciales (por ejemplo, el Motorola 6800) restringen al diseñador a conectar de esta manera los dispositivos E/S. Otros CPU comerciales (entre ellos el Intel 8085) tienen un espacio de dirección de E/S independiente; incluso así, el diseñador no se salva de asignar memoria a los dispositivos E/S. Escriba usted el código ABEL para el estado C; luego verifíquelo contra lo que sigue.

Estado C: $R/W' = 1$, $\text{cargar_IR} = 1$, $\text{cargar_AC} = 0$, $\text{cargar_PC} = 0$, ir al estado D.

En el estado D, la unidad de control inspecciona los bits del código de operación del IR y ramifica hacia uno de cuatro estados para la ejecución de una de las macroinstrucciones de la máquina. Designe los estados de la E a la H y escriba el código ABEL antes de proseguir; después verifique sus resultados contra lo que sigue.


```

module CPU_control
Title 'Unidad de control cableado'
"Unidad de control cableado de un CPU simple

Declarations
reloj, restablecer, código_up1..código_up0, AC7..AC0 PIN;
restablecer_PC, contar_PC, cargar_PC, cargar_IR, cargar_AC PIN istype 'com';
ALU_seleccionar, cargar_MAR, IR_PC, R_W PIN istpe 'com';
"ALU: seleccionar = 0 para dejar pasar y =1 para NAND
Q3..Q0 NODE istype 'reg'; salida con registro
A = [0, 0, 0, 0]; B = [0, 0, 0, 1]; "asignación de estados
C = [0, 0, 1, 0]; D = [0, 0, 1, 1];
E = [0, 1, 0, 0]; Ea = [0, 1, 0, 1];
F = [0, 1, 1, 0]; Fa = [0, 1, 1, 1];
G = [1, 0, 0, 0]; Ga = [1, 0, 0, 1];
H = [1, 0, 1, 0];

Equations
[Q3..Q0].clk = reloj;
[Q3..Q0].ar = restablecer; "restablecimiento asíncrono

state_diagram [Q3, Q2, Q1, Q0]
State A: restablecer_PC = 1; contar_PC = 0; cargar_PC = 0;
goto B; "No importa qué sucede con los otros registros
State B: restablecer = 0; cargar_PC = 0; contar_PC = 1; IR_PC = 0;
Cargar_MAR = 1; cargar_AC = 0; goto C;
State C: R_W = 1; cargar_IR = 1; cargar_AC = 0; cargar_PC = 0; goto D;
State D: if código_up1..código_up0 == [0, 0] then E
Else if código_up1..código_up0 == [0, 1] then F
Else if código_up1..código_up0 == [1, 0] then G
Else H;
State E: cargar_MAR = 1; IR_PC = 1; goto Ea;
State Ea: R_W = 1; ALU_salhab = 0; ALU_seleccionar = 0; cargar_AC = 1; goto B;
State F: R_W = 0; AC_salhab = 1; goto Fa;
State Fa: R_W = 0; AC_salhab = 1; goto B;
State G: cargar_MAR = 1; IR_PC = 1; goto GA;
State GA: R_W = 1; ALU_salva = 0; ALU_seleccionar = 1; cargar_AC = 1; goto B;
State H: if AC7..AC0 == [0, 0, 0, 0, 0, 0, 0, 0] then cargar_PC = 1; goto B;
end CPU_control

```

Figura 14. Descripción ABEL completa de la unidad de control para la trayectoria de datos de la figura 12.

Estado D: Si código_op 1..código_op == [0,0] entonces E
 Sino si código_op 1..código_op 0 == [0,1] entonces F
 Sino si código_op 1..código_op 0 == [1,0] entonces G
 Sino H

Los estados E-H especifican las microinstrucciones necesarias para ejecutar las instrucciones correspondientes. El código ABEL completo para la unidad de control se muestra en la figura 14.

Unidad de control microprogramada

Una unidad de control microprogramada utiliza un secuenciador de microprograma y una ROM para implementar los pasos de control. Cada localidad de la ROM almacena las señales de control

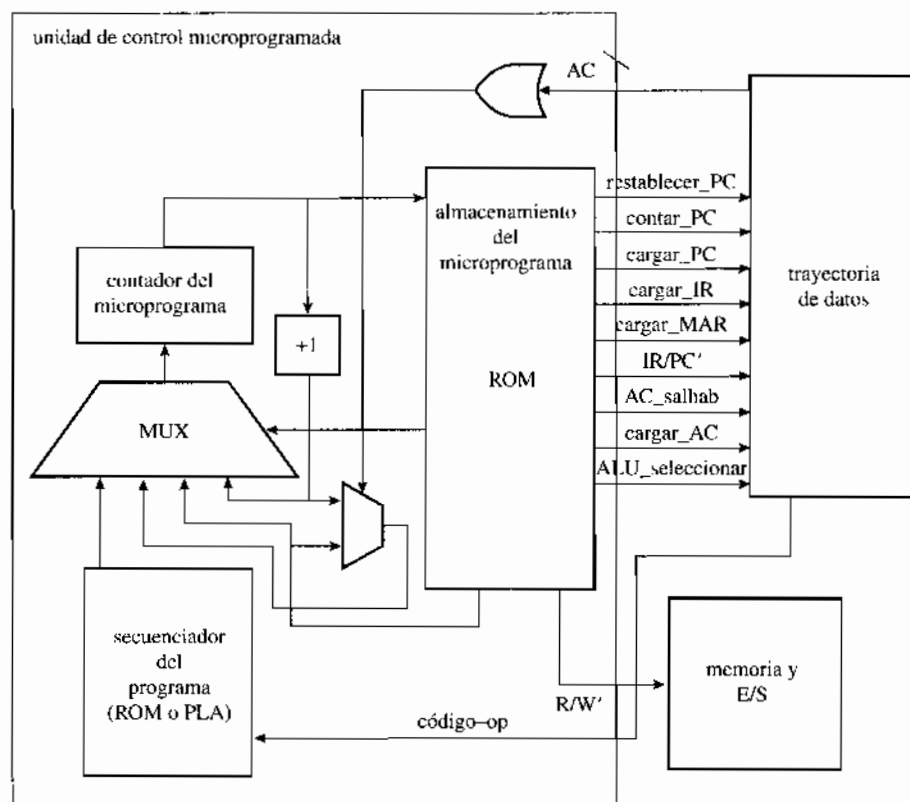


Figura 15. Diagrama de bloques de la unidad de control microprogramada.

para una sola microoperación, y el secuenciador determina cuál localidad de la ROM direccionar durante cada ciclo de reloj. Por ejemplo, la primera localidad de la ROM contiene las señales de control (microinstrucciones) requeridas para efectuar una búsqueda de instrucción. El secuenciador puede obtener su siguiente dirección a partir de la propia microinstrucción o de una tabla de traducción de direcciones. Si se conoce la siguiente dirección, entonces ésta puede codificarse en la propia microinstrucción. Éste es el caso para la búsqueda de instrucción (estado C), puesto que el paso siguiente es la decodificación de instrucción (estado D). Por otro lado, la siguiente dirección de microinstrucción después del estado D depende del código de operación de la macroinstrucción. Es posible recurrir a una tabla de consulta (otra ROM o PLA pequeña) para generar la siguiente dirección. La ventaja de la unidad de control microprogramada es que el control puede cambiarse sin rediseñar el hardware del sistema. Es factible cambiar el Firmware almacenado en la ROM de la unidad de control para implementar un nuevo esquema de control.

Tenemos la posibilidad de construir un diagrama de bloque simple de la unidad de control microprogramada. Contamos con una ROM con longitud suficiente para almacenar la totalidad de las microinstrucciones y el ancho suficiente para almacenar todas las señales de control y la información de la secuencia del microprograma. Sabemos, desde la sección anterior que disponemos de 11 estados de control; por consiguiente, el ancho de ROM más pequeño que podríamos utilizar es de 16 palabras. También necesitamos un contador de microprograma con un ancho de 4 bits.¹¹ Las entradas de estatus para la unidad de control (código_op y AC) deben

¹¹ Tal vez necesitemos unos cuantos pasos de control adicionales en comparación con la implementación de la unidad de control cableado, aunque en esta etapa de diseño podemos elegir con facilidad una ROM y un contador de microprograma más grandes.

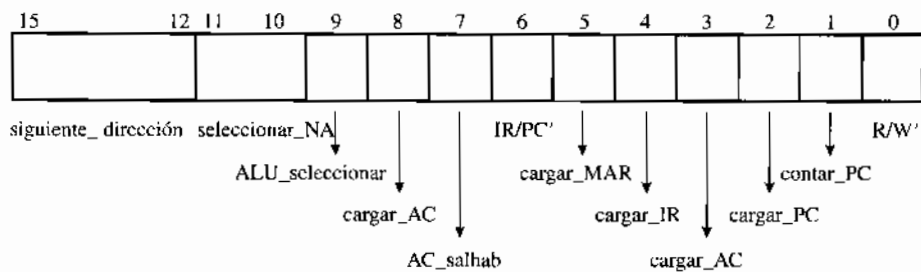


Figura 16. Formato de microinstrucciones.

procesarse por medio de hardware. Deseamos mantener flexibilidad en dicho hardware, de manera que la traducción del código_op en una dirección de microinstrucción deba hacerse en una ROM o en un PLA de manera que sea posible cambiarla sin variar el diseño del hardware.

La entrada del AC a la unidad de control se utiliza para detectar $AC = 0$. En consecuencia, es posible crear la señal de estatus mediante el paso de todos estos bits juntos por una OR. El diagrama de bloques de la unidad de control microprogramada resultante se muestra en la figura 15.

La clave para el diseño de la unidad de control microprogramada es el secuenciamiento apropiado del contador de microprograma. Hay cuatro *fuentes potenciales* de la siguiente dirección de microinstrucción.

- Si la siguiente microinstrucción está en la siguiente dirección de la ROM, entonces el contador del microprograma puede incrementarse.
- Si se requiere un salto incondicional, entonces la siguiente dirección de microinstrucción puede codificarse en la microinstrucción.
- Si la siguiente dirección de microinstrucción es condicional en el código de operación, entonces la siguiente dirección puede consultarse en una ROM o en un PLA.
- Si el código de operación es un salto condicional, en ese caso la dirección de la microinstrucción depende del contenido del AC.

Se presenta la siguiente pregunta: ¿cuántos bits son necesarios en una microinstrucción? Es posible determinar el formato de 1 microinstrucción a partir de la arquitectura del controlador microprogramado en la figura 15. Necesitamos 1 bit para cada una de las señales de control hacia la trayectoria de datos y hacia la memoria. Requerimos 2 bits para la selección de la fuente de la dirección de la siguiente microinstrucción, así como 4 bits para la dirección. Lo anterior produce un total de 16 bits en una palabra de la ROM. Asumamos que el formato de la microinstrucción es el mostrado en la figura 16. El orden de los bits no es importante, aunque los bits relacionados lógicamente (por ejemplo, bits de dirección) suelen colocarse uno junto a otro.

Estamos listos para especificar el contenido del microprograma almacenado. Almacenaremos la microinstrucción correspondiente al estado A en la primera localidad; esto permitirá al contador del microprograma apuntar a ella después de un establecimiento del hardware. ¿Cuál será el contenido de esta localidad de ROM? En este ciclo de reloj deseamos poner en cero el contador del programa ($\text{restablecer_PC} = 1$). No necesita establecerse ninguna otra de las señales de control en los bits del 9 al 0 de la microinstrucción. Es posible localizar la siguiente microinstrucción correspondiente al estado B de la sección previa en la siguiente localidad de ROM. Así, la fuente de la siguiente dirección de microinstrucción puede ser la dirección presente más 1.

La tarea siguiente es elegir la codificación para la palabra de control de 2 bits seleccionar_NA, la cual selecciona la fuente de la siguiente dirección de microinstrucción. Hacemos las asignaciones:

Seleccionar_SD = 00: Se elige la dirección presente más 1.

Seleccionar_SD = 01: Se elige el siguiente campo de direcciones de la microinstrucción.

Seleccionar_SD = 10: Se elige la dirección seleccionada por el detector AC = 0.

Seleccionar_SD = 11: Se elige la dirección del secuenciador de microprograma.

Así, para la primera microinstrucción, los bits 11 y 10 deben ser 00. El siguiente campo de direcciones de esta microinstrucción no importa, puesto que no se usa. De manera arbitraria dejaremos estos bits como 0s. Por tanto, nuestra primera microinstrucción (dirección 0 de la ROM) tiene un valor hexadecimal de 0002. (Compruébelo.)

Ejercicio 7. Determine el contenido restante del microprograma almacenado, disponiendo los estados B, C, D etc., en localidades consecutivas en la ROM. Si usted necesita más microinstrucciones que los estados que hay, inserte entonces las microinstrucciones inmediatamente después de los estados que requieren las microinstrucciones adicionales. (Emplee el código hexadecimal.)

Respuesta¹²

4 ARQUITECTURAS DE MICROPROCESADOR CONTEMPORÁNEAS

La sección anterior presentó la arquitectura de una computadora muy simple que se asemeja a algunas de las primeras computadoras electrónicas diseñadas a finales de la década de los años 40. La ENIAC (Electronic Numerical Integrator and Calculator) y la IAS (por el Institute for Advanced Study), construidas en la Universidad de Pennsylvania (1950) y de Princeton (1946), respectivamente, se encuentran entre las primeras computadoras electrónicas. Se construyeron utilizando tecnología de tubos de vacío, y cada una ocupaba una gran sala y consumía una cantidad enorme de potencia, requiriendo incluso más acondicionamiento de aire que el equipo de cómputo sólo para mantener la temperatura lo suficientemente baja a fin de que el equipo continuara operando.

El primer microprocesador de circuito integrado comercial, el 4004, que introdujo Intel Corporation en 1971, era más poderoso que la ENIAC y ocupaba un espacio más pequeño que la punta de su dedo. El microprocesador 4004 tenía 46 instrucciones, corría a una frecuencia de reloj de 108 kHz, calculaba 60,000 instrucciones por segundo y se implementó con 2,300 transistores. El 4004 marcó el principio de la revolución del microprocesador. A partir de él, la capacidad de procesamiento (velocidad y complejidad) de los microprocesadores ha crecido exponencialmente. El microprocesador Pentium II Xeon, introducido en 1998 por Intel, corre a una frecuencia de reloj de 400 MHz, calcula mil millones de instrucciones por segundo y contiene 5 millones de transistores. Esta frecuencia es 3,700 veces mayor que la del 4004, y el Pentium II calcula 16,000 veces más instrucciones por segundo. El mejoramiento en la frecuencia de reloj se debe en gran parte a los avances en la manufactura de circuitos integrados y en el diseño de circuitos transistorizados. El incremento del número de instrucciones ejecutadas más allá de la frecuencia de reloj requiere mejoras en la arquitectura del microprocesador. Los microprocesadores contemporáneos recurren a la ejecución de instrucciones en paralelo para incrementar el número de instrucciones ejecutadas por segundo. En promedio, un microprocesador contemporáneo puede ejecutar más de una instrucción por ciclo de reloj.¹³

Canalización de instrucciones

La forma más común de ejecución de instrucciones en paralelo es el *pipelining* (canalización de instrucciones). Debido a que las instrucciones se ejecutan en varios pasos pequeños (por ejemplo,

¹² Los contenidos de la ROM en localidades consecutivas que inician con la dirección 1 son 0044, 0011, 0C00, 0060, 1501, 0060, 1480, 0060, 1701, 1800, 1408.

¹³ En el año 2000, el Pentium III de Intel (y procesadores comparables de otros proveedores) habían superado bastante la capacidad del Pentium II.

	Ciclo de reloj								
	1	2	3	4	5	6	7	8	9
Búsqueda	Inst1	Inst2	Inst3	Inst4	Inst5	Inst6	Inst7	Desocupado	Desocupado
Decodificación	Desocupado	Inst1	Inst2	Inst3	Inst4	Inst5	Inst6	Inst7	Desocupado
Ejecución	Desocupado	Desocupado	Inst1	Inst2	Inst3	Inst4	Inst5	Inst6	Inst7

Figura 17. Tabla de reservaciones ejemplo para una arquitectura en pipeline.

búsqueda, decodificación, ejecución), es posible que se traslapen los pasos de instrucciones diferentes.

Mientras se ejecuta una instrucción, la siguiente se está decodificando y la posterior buscando en la memoria. Una vez que el pipeline está lleno, una instrucción termina de ejecutarse cada ciclo de reloj.

La ejecución de instrucciones en pipeline puede analizarse utilizando una *tabla de reservaciones*, la cual identifica los recursos de hardware ocupados por una instrucción en un tiempo determinado.

Considere una arquitectura donde las etapas de búsqueda, decodificación y ejecución son unidades de hardware separadas que pueden operar sobre diferentes instrucciones al mismo tiempo. Si las instrucciones de la 1 a la 7 requieren cada una un ciclo de reloj para la búsqueda, decodificación y ejecución, entonces es posible representar su ejecución mediante la tabla de reservaciones que se presenta en la figura 17.

Cuando se inicie la ejecución, las unidades de decodificación y ejecución no tienen ningún trabajo que hacer (están desocupadas). Una vez que el pipeline se "llena" (ciclo de reloj 3), una instrucción termina de ejecutarse en cada ciclo de reloj. Con la arquitectura en pipeline se ejecutan 7 instrucciones en 9 ciclos de reloj. Si la arquitectura no está en pipeline, se requieren entonces 21 ciclos de reloj para el mismo número de instrucciones.

Es probable que una o más instrucciones requerirán más de un ciclo de reloj en la etapa de ejecución. En esta etapa únicamente, por ejemplo, una multiplicación o división puede tardar 10 o más ciclos de reloj; una suma quizá tarde sólo 1 ciclo de reloj. ¿Cómo sería la tabla de reservaciones cuando se ejecuta una instrucción de este tipo, digamos una multiplicación? La siguiente instrucción debe esperar a que la multiplicación termine. Si la instrucción 3 en la figura 17 es una multiplicación y requiere 2 ciclos de reloj para la ejecución, entonces la tabla de reservaciones sería como la que se observa en la figura 18.

La instrucción 4 no puede utilizar la unidad de ejecución hasta que haya terminado la instrucción 3. En consecuencia, las unidades de búsqueda y decodificación, están desocupadas durante el ciclo de reloj 6, y el tiempo total de ejecución para las siete instrucciones requiere un ciclo de reloj adicional.

Ejercicio 8. Elabore nuevamente la tabla de reservaciones de la figura 17 suponiendo que la instrucción 2 necesita buscar datos en la memoria para completar su ejecución.

Respuesta¹⁴

La regularidad de un pipeline también puede alterarse por causa de instrucciones de salto condicionales. Cuando se extrae de la memoria de programa el código de operación correspondiente a un salto condicional (por ejemplo, saltar si el resultado de una operación es cero), la

¹⁴ Puesto que sólo hay una unidad de hardware para la extracción de instrucciones de la memoria, la búsqueda de la instrucción 4 sufre el retraso debido a la búsqueda de datos de la instrucción 2. Advierta que la instrucción 3 se busca en el ciclo 3, pues no hay manera de conocer que la instrucción 2 requiera una búsqueda de datos hasta después de decodificarla.

	Ciclo de reloj									
	1	2	3	4	5	6	7	8	9	10
Búsqueda	Inst1	Inst2	Inst3	Inst4	Inst5	Desocupado	Inst6	Inst7	Desocupado	Desocupado
Decodificación	Desocupado	Inst1	Inst2	Inst3	Inst4	Desocupado	Inst5	Inst6	Inst7	Desocupado
Ejecución	Desocupado	Desocupado	Inst1	Inst2	Inst3	Inst3	Inst4	Inst5	Inst6	Inst7

Figura 18. Tabla de reservaciones donde una instrucción en la etapa de ejecución requiere dos ciclos de reloj.

siguiente instrucción por ejecutarse no se conoce hasta después de que se ejecuta el salto. Para mantener el pipeline lleno, la instrucción siguiente debe buscarse en la memoria mientras se decodifica la bifurcación. ¿Cómo sabe el procesador qué instrucción buscar? Existen dos posibilidades para la instrucción siguiente, y los microprocesadores más actuales adivinan casi cuál es la correcta y luego la eligen. Si la suposición es equivocada, es necesario purgar el pipeline y buscar la instrucción correcta en la memoria.

Unidades de hardware en paralelo

En la sección anterior, presentamos tres ejemplos que reducen la eficacia del pipeline: ciclos múltiples para ejecución, búsquedas múltiples por instrucción y saltos. Además, la tasa más rápida de ejecución utilizando el pipeline no puede exceder de una instrucción por ciclo de reloj.¹⁵ ¿Cómo puede un microprocesador ejecutar más de una instrucción por ciclo de reloj? Tiene que buscar múltiples instrucciones al mismo tiempo, y también decodificarlas y ejecutarlas en paralelo. Cada trayectoria en paralelo debe ponerse en pipeline para maximizar el rendimiento.

Los microprocesadores más modernos tienen dos o más pipelines de instrucciones, por lo que pueden ejecutar en promedio más de una instrucción por ciclo de reloj.¹⁶ Con frecuencia se emplean pipelines de instrucciones múltiples para ejecutarlas en paralelo. De acuerdo con lo que usted conoce acerca de la programación, los lenguajes de ésta son secuenciales y las sentencias en un programa se ejecutan de esa misma manera. Sin embargo, muchas sentencias en un programa típico se ejecutan en paralelo.

Considere dos sentencias de programa, una que suma los valores almacenados en las variables *A* y *B*, y que almacena el resultado en *X*, y otro que suma los valores almacenados en las variables *C* y *D*, y almacena el resultado en *Y*. El orden de ejecución de estas dos declaraciones no marca ninguna diferencia para el resultado del cálculo. ¿Es posible ejecutar las sentencias en paralelo? Desde luego que es posible, ¿por qué no? Suponga que la segunda sentencia suma *C* y *X*, en lugar de *C* y *D*. ¿En este caso es factible ejecutar en paralelo la sentencia? Puesto que la primera sentencia calcula el valor de *X*, la ejecución de la misma debe completarse antes de que pueda ejecutarse la segunda (la segunda requiere el resultado de la primera.) Lo anterior se conoce como una *dependencia de datos* entre sentencias (o instrucciones). Los compiladores contemporáneos son capaces de identificar sentencias que es posible ejecutar en paralelo. Algunas veces es necesario reordenar las sentencias en un programa para maximizar la cantidad de cómputo en paralelo.

¹⁵ Un pipeline siempre debe llenarse antes de que pueda producir resultados útiles. De tal modo, el número de ciclos de reloj para ejecutar *n* instrucciones puede ser a lo más *n* más el número de ciclos de reloj que se requieren para llenar el pipeline.

¹⁶ *Idem*.

EJEMPLO 1

Suponga que existe un lazo que suma los elementos de dos vectores, A y B, y que se almacena el resultado en un tercer vector, C. Las declaraciones del programa se asemejan a lo siguiente:

```
Para i en el ciclo de 0 a 99
  C[i] = A [i] + B[i];
fin ciclo;
```

Supóngase ahora que hay un procesador con dos pipelines de instrucciones (en lugar de uno) para ejecutar el código. Puesto que todas las adiciones son independientes, este procesador sería dos veces más rápido que el original. ■

Otro uso de los pipelines de instrucciones en paralelo es la ejecución de ambas trayectorias posibles en una instrucción de salto condicional. Puesto que se están ejecutando ambas trayectorias, no se pierde tiempo por suposiciones equivocadas.

Jerarquía de memoria

En la figura 13 se muestra un diagrama de bloques simplificado de un sistema de computadora. La memoria en esta figura se presenta como un bloque, aunque en los sistemas de computadora hoy día la memoria viene en varios bloques de tamaño y velocidad variable. En la computadora personal actual resultan comunes 64 o 128 MB de RAM. Sin embargo, la RAM o memoria principal en un sistema de computadora, es lenta y se requieren varios ciclos de reloj del CPU para tener acceso a los datos en ella. Es común contar con una memoria de alta velocidad entre el CPU y la memoria principal, de manera que los datos puedan tener acceso con rapidez. Esta memoria de alta velocidad recibe el nombre de *memoria caché*. La memoria caché es costosa en comparación con la principal, por lo que el tamaño de la misma por lo común es limitado (casi siempre 1 MB o menos). El sistema intenta mantener las instrucciones y los datos utilizados con mayor frecuencia en la memoria caché, optimizando de esa manera el desempeño del sistema.

Cuando los datos que se requieren no están en la caché (lo que se denomina un *fallo de caché*), deben obtenerse entonces de la memoria principal. La interfaz entre la caché y la memoria principal se diseña a menudo de modo tal que grandes segmentos de memoria, denominados páginas, pueden moverse de la principal a la caché cuando ocurre la falta de esta última.

Los microprocesadores de nuestros días cuentan con memorias caché independientes para instrucciones y para datos, y éstas se integran por lo general en el CI del microprocesador. Muchos sistemas de computadora tienen dos niveles de caché entre el CPU y la memoria principal, con el segundo nivel fuera del CI (no integrado con el microprocesador). La ventaja de dos niveles de caché es que el segundo puede ser más grande que el primero debido a que no está en el CI del microprocesador. Sin embargo, también es más lento que el primer nivel por la misma razón.

Computadora de conjunto complejo de instrucciones (CISC)

Los microprocesadores más contemporáneos se basan en una de dos arquitecturas: computadora de conjunto complejo de instrucciones (CISC) y computadora de conjunto reducido de instrucciones (RISC).

Una arquitectura CISC puede tener unos cuantos cientos de instrucciones y el hardware incluye muchas características para la ejecución directa de instrucciones complejas. Los microprocesadores basados en la arquitectura CISC prevalecieron a principios y mediados de la década de los ochenta y siguen encontrándose en los inicios de este siglo. La familia de microprocesa-

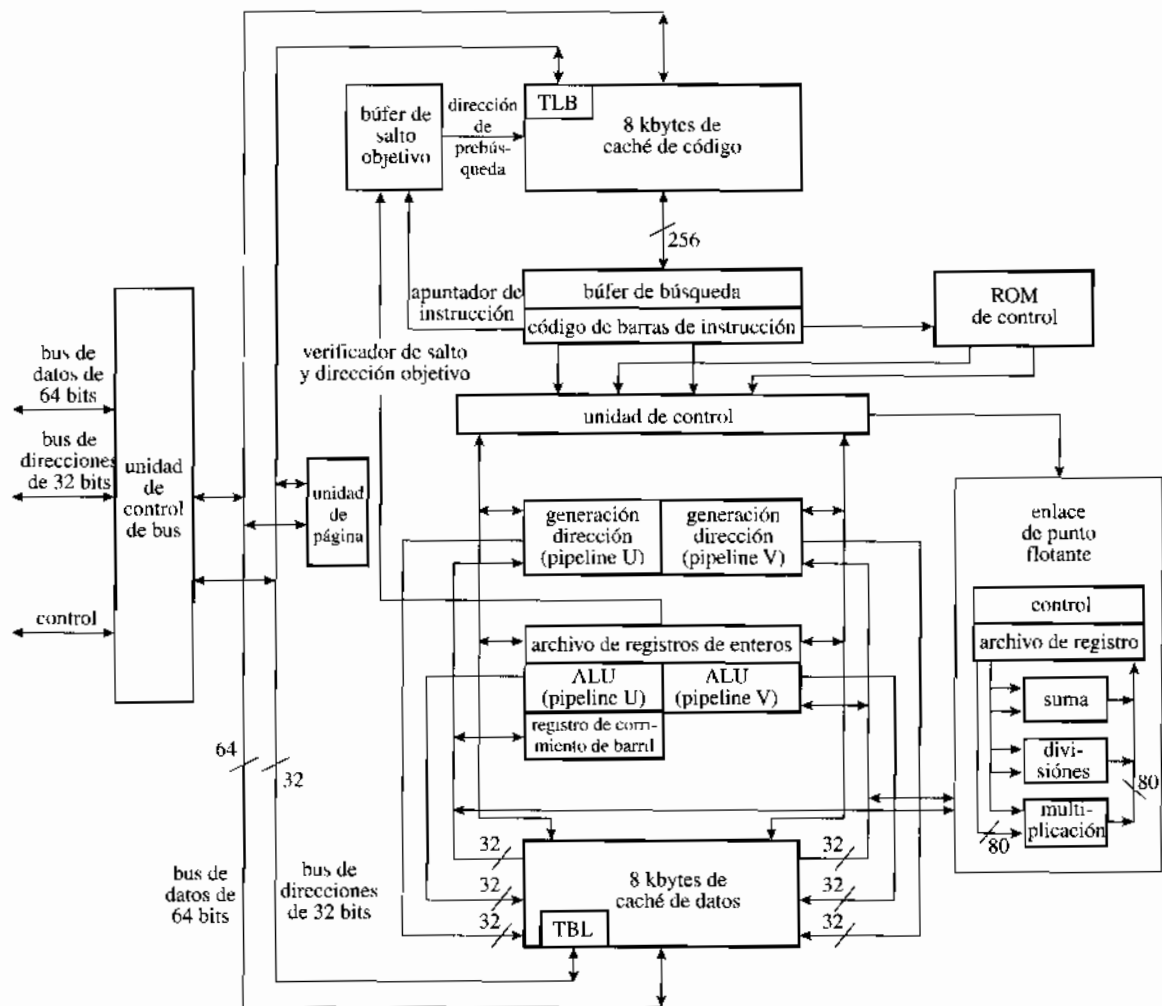


Figura 19. Diagrama de bloques de un microprocesador Pentium. (© Intel Corporation, reproducido con autorización)

dores x86 de Intel Corporation (que incluye la serie Pentium) se basa en una arquitectura CISC. El diagrama de bloques de un microprocesador Pentium se muestra en la figura 19.

El microprocesador Pentium tiene un bus de direcciones de 32 bits y un bus de datos de 64 bits. La mayor parte de las instrucciones son de 32 bits, por lo que casi todo el tiempo el microprocesador recibe dos instrucciones con una búsqueda de instrucciones en memoria. Hay dos pipelines de instrucciones (U y V), de modo que ambas instrucciones pueden decodificarse y ejecutarse al mismo tiempo (siempre y cuando una no dependa de la otra). El compilador de lenguaje de alto nivel debe realizar un buen trabajo al organizar el código ensamblador para permitir el uso máximo del procesamiento de instrucciones en paralelo. El Pentium tiene cachés separadas de instrucción (código) y datos, cada una de 8 kbytes. Hay una unidad de punto flotante independiente de los dos pipelines de instrucciones para manejar operaciones de punto flotante difíciles. Las operaciones de multiplicación y división de punto fijo (enteros) también las maneja la unidad de punto flotante. Advierta las unidades de generación de direcciones en el pipeline de instrucciones. En una arquitectura CISC es común que las operaciones aritméticas impliquen datos contenidos en memoria. Después de decodificar la instrucción, la dirección de cualquier operando de la instrucción debe determinarse (o generarse).

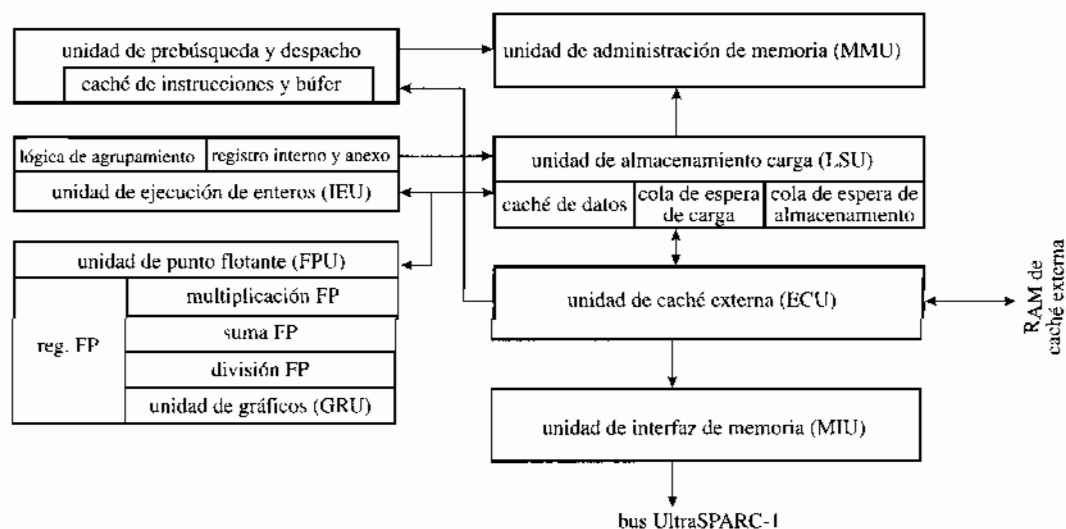


Figura 20. Diagrama de bloques del microprocesador UltraSPARC RISC. (© 1997 SUN Microsystems, Inc. Derechos reservados. Utilizado con autorización.)

Computadora de conjunto de instrucciones reducidas (RISC)

La investigación llevada a cabo en la década de los ochenta reveló que casi todos los programas utilizaban sólo un pequeño porcentaje (alrededor de 20%) de las instrucciones de un microprocesador CISC la mayor parte del tiempo (casi 90 por ciento).

La unidad de control podría simplificarse de manera considerable al minimizar el número de instrucciones, y dicha simplificación permitiría velocidades de reloj mucho más altas. Además, la simplificación de la unidad de control significa espacio adicional en el CI para los pipelines de instrucciones en paralelo, registros de datos y memoria caché.

Los microprocesadores RISC aparecieron en el mercado al final de los años ochenta y son dominantes hoy día en las estaciones de trabajo basadas en UNIX. Los ejemplos incluyen el MIPS R4000, el SPARC de Sun Microsystems, el Alpha de Digital Equipment Corporation, el RS/6000 de IBM, el Intel i860/960, el Motorola 88000 y el PA RISC de Hewlett Packard. Los primeros procesadores RISC se distinguían de los CISC por pipelines de instrucciones y cachés de datos e instrucciones independientes, además de un conjunto de instrucciones pequeño. En la actualidad, la frontera entre las arquitecturas RISC y CISC es poco clara, aunque la diferencia en la complejidad del conjunto de instrucciones sigue existiendo. Similares a las actuales arquitecturas RISC, la mayor parte de las arquitecturas CISC modernas tienen pipelines de instrucciones múltiples y cachés de datos e instrucciones independientes.

En la figura 20 se presenta el diagrama de bloques del microprocesador UltraSPARC de Sun Microsystems. Éste soporta palabras de datos de 64 bits, cuenta con una dirección física de 41 bits, y tiene una ejecución pico de cuatro instrucciones por ciclo de reloj con una frecuencia de reloj máxima de 250 MHz. El UltraSPARC puede mantener la tasa pico de cuatro instrucciones por ciclo incluso en la presencia de instrucciones de salto condicional. La tasa de ejecución se retarda sólo por causa de dependencias de datos entre las operaciones. Dispone de cachés de instrucciones y datos de 16 kbytes, dos ALU para números enteros y unidades de ejecución de operaciones de punto flotante y gráficos. Todos los accesos de memoria para los datos se manejan mediante instrucciones de carga/almacenamiento.

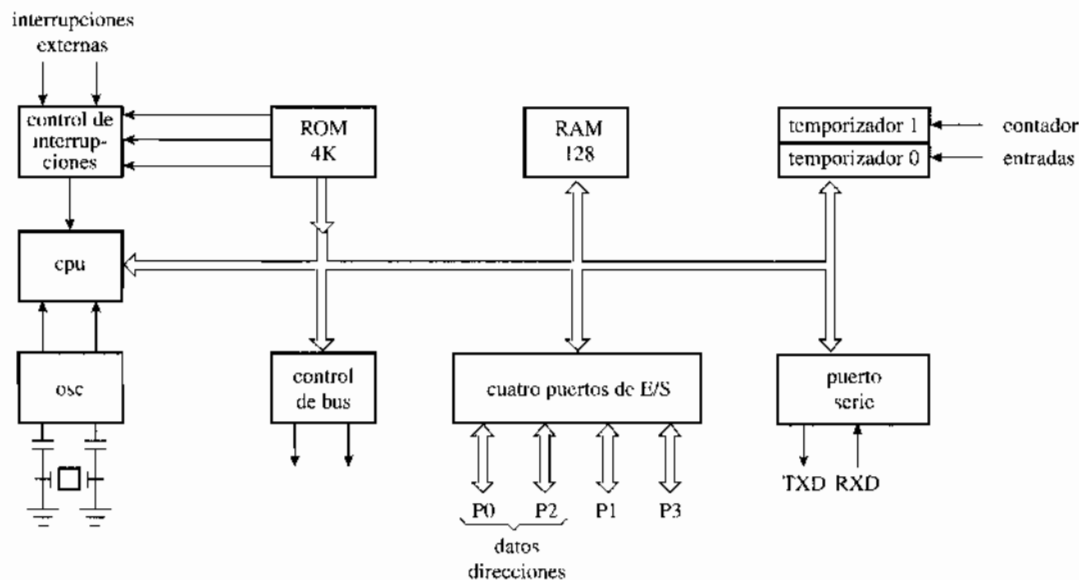


Figura 21. El diagrama de bloques del microcontrolador 8051. (© Philips Semiconductors, reproducida con autorización.)

5 ARQUITECTURAS DE MICROCONTROLADOR

Las arquitecturas de los microcontroladores y de los microprocesadores difieren en las formas que dictan sus aplicaciones propuestas. Los microcontroladores se diseñan para aplicaciones en tiempo real, aquellas en las cuales resulta crítico que el sistema responda en tiempo real. Un ejemplo de un sistema en tiempo real es el correspondiente a un radar de aviones, el cual debe responder a objetos en su espacio de sensores para que el piloto pueda reaccionar de manera apropiada. Por otro lado, una computadora personal no es de tiempo real, pues la única consecuencia de una computadora lenta es la frustración del usuario. Otros ejemplos de sistemas en tiempo real son dispositivos de comunicación, como los de voz de dos vías (teléfonos) y los dispositivos de audio y video de una vía (TV y radio.) Si se interrumpe la señal de audio y video de este tipo de sistemas, esa situación inutiliza al sistema.

Los microcontroladores tienen características específicas para la implementación de sistemas en tiempo real. Es común que incluyan memoria integrada programable sólo de lectura (PROM), convertidores analógico-digital e interrupciones de alta prioridad. Una interrupción es una señal de hardware o software que ocasiona que la secuencia de instrucciones salte hacia una localidad específica en la memoria donde existen instrucciones especiales. Estas últimas se diseñan para servir a una petición particular del sistema. Por ejemplo, un sistema de control que supervisa la temperatura de un horno quizá se interrumpa si la temperatura excede de un nivel crítico. La respuesta del sistema tal vez sea sacar de servicio el horno de manera inmediata para evitar las consecuencias severas de una temperatura elevada.

La PROM integrada es útil para codificar el arranque del sistema, y en algunas aplicaciones puede contener todo el código de instrucciones. Los convertidores analógico-digitales son convenientes para la conversión de señales analógicas de sensores como los de temperatura. El diagrama de bloques del microcontrolador Intel 8051 se muestra en la figura 21.

Hay varios bloques en un microcontrolador que lo distinguen de un microprocesador: el controlador de interrupciones, la ROM, la RAM, los puertos de E/S y un puerto serie. Es viable que la ROM y la RAM internas se usen en el código crítico y en los segmentos de datos que requieren

operación muy rápida para promover una restricción en tiempo real. El controlador de interrupciones maneja varias interrupciones con diferentes prioridades, para determinar cuál petición debe manejarse y en qué orden. Por ejemplo, en una fábrica, a una señal de interrupción de un sensor de monóxido de carbono se le asignaría una prioridad superior que a una señal de interrupción de un reloj que marca la hora del día y se usa para ajustar las especificaciones del termostato en la instalación. Los microcontroladores varían de manera significativa en la capacidad de cómputo; se encuentran en el mercado desde microcontroladores de 8 bits para aplicaciones de bajo costo (control de clima doméstico) hasta microcontroladores de 32 bits para aplicaciones de alta demanda (procesamiento de señales de radar). En contraste, no hay muchas aplicaciones hoy día para un microprocesador de 8 bits, puesto que se espera que todas las computadoras de propósito general funcionen con sistemas y software avanzados.

RESUMEN Y REPASO DEL CAPÍTULO

El tema de la arquitectura de computadoras es vasto y requiere varios semestres de estudio para la profundidad y extensión que requiere un ingeniero en cómputo. El fin, aquí es establecer un puente entre el diseño de circuitos digitales y el diseño de computadoras a gran escala, así como proporcionar un excelente cimiento para el estudio de la arquitectura de computadoras. Se abarcaron los siguientes temas:

- Unidades de control y de trayectoria de datos de un procesador.
- Diseño de un procesador simple: multiplicador en serie.
- Computadora básica de programa almacenado.
- Ciclo de búsqueda, decodificación y ejecución.
- Unidad de procesamiento central (CPU).
- Diseño de un CPU simple.
- Notación de lenguaje de transferencia de registros (RTL).
- Implementaciones de la unidad de control.
 - Unidad de control cableada.
 - Unidad de control *microprogramada*.
- Arquitecturas de microprocesador contemporáneas.
- Pipelining de instrucciones.
- Unidades de hardware en paralelo.
- Jerarquía de memoria.
- Computadora de conjunto complejo de instrucciones (CISC).
- Computadora de conjunto reducido de instrucciones (RISC).
- Arquitecturas de microcontrolador.

PROBLEMAS

1. Rediseñe la trayectoria de datos del multiplicador de suma y corrimiento de la sección 1 de manera que las operaciones de suma y corrimiento se ejecuten en un ciclo de reloj.
2. Rediseñe la unidad de control del multiplicador de suma y corrimiento de manera tal que si el bit multiplicador es 0, la unidad de procesamiento salta completamente el paso de suma. Esto es, la trayectoria de datos efectúa dos operaciones de corrimiento en ciclos de reloj consecutivos.
3. Escriba una descripción ABEL del multiplicador de suma y corrimiento descrito en la sección 1.
4. Se necesita un procesador que calcule la diferencia entre dos entradas sucesivas de 8 bits a un sistema. El procesador recibe una secuencia de valores de 8 bits, $x(t)$, y produce el resultado $y(t) = x(t-1) - x(t)$. Esto es, la salida presente es la entrada previa menos la entrada presente. Debe producirse una salida cada ciclo de reloj. Escriba una descripción ABEL para este procesador.

5. Diseñe un procesador que acepte una secuencia continua de valores de 8 bits, y calcule el promedio de los ocho valores más recientes. La salida es $y(t) = [x(t) + x(t-1) + \dots + x(t-7)]/8$. Escriba una descripción ABEL del procesador.
6. Combine los procesadores de los problemas 4 y 5 para crear un sistema con una salida de un bit que se asevera con 1 lógico, siempre que la diferencia entre las dos entradas más recientes supere la mitad del promedio de las ocho entradas más recientes.
7. Diseñe las unidades de control y de trayectoria de datos de un procesador que acumula cuatro números de 8 bits que se proporcionan de manera secuencial. En cuatro ciclos de reloj consecutivos, se proporcionan cuatro números de 8 bits a las entradas y el procesador debe acumular sus valores en un registro. El procesador automáticamente inicia de nuevo después de que se recibe la última de las cuatro entradas. Dibuje un diagrama de bloques de la unidad de trayectoria de datos y escriba descripciones ABEL de las unidades de control y de trayectoria de datos del procesador.
8. La función $y(t) = x(t) + 5x(t-1) + 2x(t-2)$ necesita calcularse para la implementación de un filtro digital. Diseñe un procesador que implemente esta función y escriba su descripción ABEL. ¿Conviene utilizar un multiplicador para implementar esta función?
9. Especifique las instrucciones y trayectoria de datos para un CPU que tiene sólo tres instrucciones y puede utilizarse para implementar programas arbitrarios. (Considere el ejemplo de la sección 2 y determine cuál de las cuatro instrucciones es redundante.)
10. Haga una descripción ABEL del CPU simple de la figura 9; incluya tanto la unidad de trayectoria de datos como la de control.
11. Respecto a la descripción ABEL de la figura 14, describa cómo un compilador ABEL optimiza las expresiones de salida. Para varios estados, algunas salidas no están definidas explícitamente. ¿Cómo considera a éstas la herramienta de síntesis ABEL? Rescriba las descripciones de manera que la herramienta de síntesis ABEL aproveche al máximo las condiciones irrelevantes.
12. Diseñe un procesador de 8 bits que ejecute la operación aritmética suma, y las operaciones lógicas AND, OR-Exclusiva y NOT. Suponga que el código de instrucciones y los valores de los datos de entrada se almacenan en registros, y que el resultado de la operación se almacena en un cuarto registro. Realice una descripción ABEL del procesador. Dibuje un diagrama de bloques de sus unidades de trayectoria de datos y control. Sintetice la descripción ABEL utilizando el software de Xilinx.
13. ¿Qué determina la máxima frecuencia de reloj de un microprocesador? (Quizá haya más de un factor. Considere todos.)
14. El número máximo de instrucciones que es posible ejecutar en un solo ciclo de reloj se determina mediante la arquitectura del microprocesador. Sin embargo, los rendimientos de varios segmentos de la arquitectura tienen que igualarse para poder lograrlo. Explique lo que esto significa. Proporcione un ejemplo en el cual un sistema tiene trayectorias de ejecución paralelas, aunque el número máximo de instrucciones ejecutadas por ciclo sea limitado.
15. Para cada uno de los siguientes pares de sentencias de programa, determine si éstas tienen o no dependencias de datos. En otras palabras, ¿es posible cambiar su orden sin afectar el resultado de la computación?
 - a. $A = B + C, A = C - D$
 - b. $A = B + C, D = B + E$
 - c. $A = B + C, B = D + C$
 - d. $A = B + C, D = A + C$
16. La siguiente secuencia de sentencias de lenguaje de alto nivel se ejecutará en un microprocesador que cuenta con dos pipelines de instrucciones en paralelo. Reordene las declaraciones de manera que se requiera el número más corto de ciclos para su ejecución y se consiga el mismo resultado de cómputo. Suponga que cada instrucción requiere el mismo número de ciclos para la búsqueda, decodificación y ejecución.

$$W = A + B$$

$$X = W + D$$

$$Y = Z + C$$

$$Z = A + D$$

17. Para la respuesta del problema 16, dibuje una tabla de reservaciones para los dos pipelines de instrucciones del CPU. Suponga que todas las variables son localidades de registro, por lo que no se necesitan accesos de memoria para obtener los datos, y considere que la búsqueda, decodificación y ejecución requiere en total un ciclo de reloj. ¿Cuántos ciclos de reloj se ahorran reordenando las sentencias?
18. ¿Qué intervalo de valores puede representarse en el formato de complemento a dos, por medio de los microprocesadores Pentium y UltraSPARC?
19. ¿Cuántas localidades de memoria se direccionan directamente por medio del microprocesador Pentium? ¿Cuántas mediante el microprocesador UltraSPARC?
20. En promedio, ¿cuál microprocesador puede ejecutar más instrucciones por segundo, un Pentium a 400MHz o un UltraSPARC a 250MHz?
21. Describa una interfaz asíncrona entre un procesador y un sistema de memoria. Dibuje los diagramas de estados que caractericen el control en cada parte (procesador y memoria) del sistema.

MOSFET y transistores de unión bipolar

Este apéndice constituye una breve explicación de los fundamentos de algunos aspectos de la electrónica, suficientes para trabajar con la terminología y la operación general de los circuitos lógicos electrónicos. La aquí expuesta podría ayudarle a comprender la terminología de los manuales de datos de los fabricantes de circuitos lógicos.

El transistor, el bloque constitutivo a partir del cual se construyen todas las compuertas lógicas, se inventó hace medio siglo. Está hecho de materiales *semiconductores* que, como su nombre lo indica, tienen propiedades de conductividad eléctrica que se encuentran entre las de los buenos conductores (cobre) y las de los aisladores (caucho). El *silicio* y el *germanio* son ejemplos de semiconductores, y ambos se encuentran en la cuarta columna de la tabla periódica, por lo cual tienen una valencia de 4. El silicio es el semiconductor que más se utiliza en la actualidad.

La conductividad de semiconductores puros puede modificarse y controlarse introduciendo átomos de materiales de las columnas adyacentes de la tabla periódica en su estructura cristalina. Esto se conoce como *impurificación*. La impurificación del silicio con átomos de valencia 5 (por ejemplo, antimonio, arsénico) aumentará el número de portadores de carga negativa (electrones). El resultado se denomina semiconductor *tipo n*. La impurificación con átomos de valencia 3 (boro, galio, entre otros) incrementará el número de portadores de carga positiva (hueco), dando lugar a un semiconductor *tipo p*. El material como un todo es eléctricamente neutro en ambos casos.

Una *unión pn* se forma encarando un semiconductor tipo *p* con un tipo *n*, como se indica en forma estilizada en la figura 1.¹ La *polarización* a través de la unión se relaciona con el voltaje que resulta de las conexiones externas. Si este voltaje es positivo del lado *p* al *n*, se dice que la unión estará *polarizada directamente*; si es negativo, estará *polarizado inversamente*.

Un dispositivo consistente en una unión *pn* recibe el nombre de *diodo*. La corriente (directa) en un diodo polarizado directamente es muy baja hasta que el voltaje alcanza cerca de 0.6 V; ésta aumenta después muy rápido aunque casi de manera lineal con un incremento adicional en el voltaje, como si el diodo fuera un resistor de bajo valor. Cuando el diodo está polarizado inversamente, la corriente *inversa* (o *de fuga*) es en extremo baja, varios órdenes de magnitud inferior que la corriente directa. (Si la corriente directa está en mA, la corriente inversa estará en μA o en valores inferiores.) Sin embargo, si el voltaje inverso se incrementa hasta el valor de *rompimiento inverso* (los valores típicos corresponden a 40-50V), la corriente inversa aumenta de manera precipitada, de ahí el término *voltaje de ruptura*. (Hay aplicaciones en las cuales se utiliza esta propiedad, pero no en circuitos lógicos.)

¹ En la construcción real, la unión se forma sobre un sustrato de silicio distribuyendo la impurificación de los átomos de impurezas de manera que se cree un cambio abrupto en las densidades de portadores.

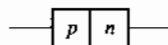


Figura 1. Diodo semiconductor.

La única forma de controlar la corriente en un diodo es por medio del voltaje a través de la unión. Sería muy útil que existieran otros medios de controlar la corriente. Una manera de conseguirlo es formar una combinación de dos uniones, es “emparedado”: *pnp* o *npn*, y puede haber varios caminos para conseguir lo anterior en un circuito integrado.

El paso inicial es impurificar una “rebanada” de silicio para formar un agregado de material tipo *n* o tipo *p*. Después de eso, distintos procedimientos dan lugar a diferentes variedades: cada una de éstas lleva el nombre genérico de *transistor*.

Varios tipos de transistores han aparecido a lo largo del tiempo; algunos de ellos se han descartado completamente al arribar variedades basadas en principios ligeramente diferentes. Otros aún se usan en aplicaciones distintas de circuitos lógicos. Aquí se explicarán dos variedades.

MOSFET

En la actualidad se usan dos tipos de transistores en los dispositivos lógicos. Uno es un MOSFET (MOS por las siglas en inglés de metal-oxide semiconductor; y FET, por las de transistor de efecto de campo). Son dos variedades de MOSFET, la versión en *modo de acrecentamiento* es la única que se usa bastante en circuitos lógicos, por lo que es la que más se describirá aquí.² En la figura 2 puede observarse una versión estilizada como una sección transversal. Lo que se designa como masa (bulk) también recibe el nombre de *sustrato*.

Aunque sólo se muestran tres terminales externas en la figura 2, el sustrato constituye otra terminal que casi siempre se conecta a tierra en las aplicaciones de circuitos digitales. Esto es, el MOSFET en modo de acrecentamiento es un dispositivo de cuatro terminales. La terminal de compuerta —un conductor— está separada del resto del dispositivo por un aislador (indicado por medio del rayado). Eso significa que no hay conducción de corriente desde la compuerta hasta cualquier otra parte del MOSFET.³ La región *p* que se encuentra entre las dos regiones *n* (drenaje y fuente) se denomina *canal*, debido a que es la trayectoria en la cual la corriente fluye bajo voltajes de polarización adecuados. Durante la conducción de corriente se invierte la concentración de portadores en el canal: los electrones son atraídos ahí por el campo eléctrico inducido mediante el voltaje positivo de compuerta a fuente. Por tanto, el transistor en la figura 2 se denomina MOSFET de *canal n* o *tipo n*. Cuando el voltaje de compuerta a fuente es 0, no hay co-

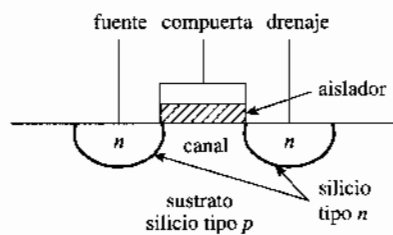


Figura 2. Sección transversal del transistor MOSFET en modo de acrecentamiento.

² La otra versión es el MOSFET en *modo de agotamiento*.

³ Estrictamente hablando, existe cierta corriente (capacitiva), aunque ésta es lo suficientemente pequeña para asignarle un valor cero.

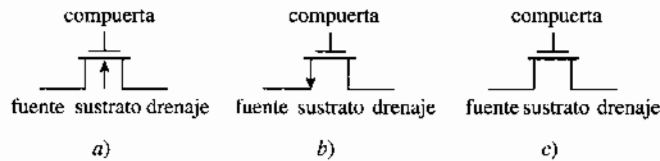


Figura 3. Símbolos de circuito comunes para el MOSFET de canal n .

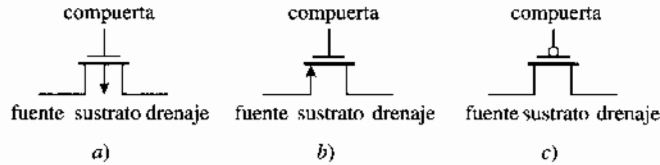


Figura 4. Símbolos de circuito comunes para el MOSFET de canal p .

riente en el canal. Sin embargo, para un voltaje suficientemente positivo de compuerta a fuente, existe un flujo de corriente del drenaje a la fuente que no depende del valor preciso del voltaje de drenaje a fuente. En estas circunstancias el MOSFET actúa como un interruptor; es así como se usa en las aplicaciones de circuitos lógicos.⁴ En la figura 3 se indican símbolos comunes que se emplean para el MOSFET en los circuitos.

El símbolo más genérico es el de la figura 3a. En los circuitos lógicos, puesto que el sustrato se conecta casi siempre a tierra,⁵ a menudo se omite la terminal de éste, como en la figura 3b. Además, en virtud de que el MOSFET es un dispositivo físicamente simétrico con respecto a las terminales de la fuente y el drenaje, es común utilizar el símbolo como en la figura 3c, donde se elimina la flecha y no se distinguen las terminales de la fuente y el drenaje, ¡salvo por sus leyendas!

Si se intercambian las regiones p y n , se forma una variación de la estructura en la figura 2. El resultado se denomina MOSFET de canal p o tipo p . En este caso es un voltaje negativo de compuerta a fuente que resulta en una corriente de fuente a drenaje.

Los símbolos de circuito para el MOSFET de canal p se muestran en la figura 4. La única distinción corresponde a la dirección de las flechas en las primeras dos partes y a la burbuja en la compuerta en la figura 4c en comparación con la figura 3c.

De acuerdo con la descripción anterior, es evidente que, utilizando los voltajes apropiados de compuerta a fuente, el MOSFET se comporta como un interruptor controlado por voltaje: un voltaje positivo de compuerta a fuente cierra el interruptor en un tipo n y un voltaje negativo lo hace en un tipo p . En ambos tipos, existe una corriente despreciable en la compuerta en ambas condiciones. Entre la fuente y el drenaje hay: a) una impedancia muy baja (idealmente cero) cuando el transistor conduce (el interruptor está *activado*), y b) una impedancia muy alta (idealmente infinita) cuando el transistor no conduce (el interruptor está *desactivado*).

Transistor de unión bipolar

Otro método para formar dos uniones, una al lado de la otra se ilustra en la figura 5. Este arreglo recibe el nombre de *transistor de unión bipolar* (BJT). Esta versión es un transistor *npn*. El ancho físico de la base es muy pequeño. La región del emisor se impurifica en mayor grado que

⁴ Para voltajes inferiores, la corriente es proporcional al voltaje; en este caso el dispositivo actúa como un resistor. Este modo de operación resulta útil en otras aplicaciones, pero no en circuitos lógicos.

⁵ El sustrato se conecta siempre al voltaje de suministro más bajo en el circuito; éste por lo general corresponde a tierra.

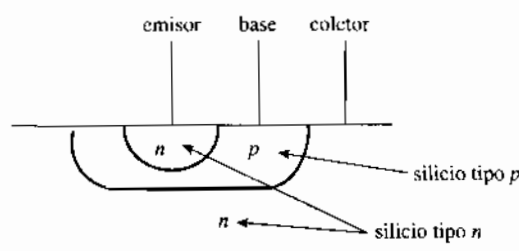


Figura 5. Sección transversal de un transistor de unión bipolar.

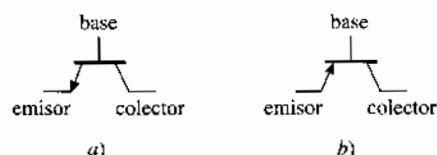


Figura 6. Símbolos de circuito para transistores bipolares. a) Transistor npn. b) Transistor pnp.

la región *n* del sustrato, en tanto que la base se impurifica un poco. En los circuitos lógicos también es necesario que el transistor opere más o menos como un interruptor. Si ambas uniones se polarizan inversamente, se dice que el transistor está en la región *de corte*. Esto requiere que el voltaje base-emisor (V_{BE}), así como el voltaje base-colector (V_{BC}) sean negativos. En estas condiciones no fluirá corriente (en realidad, muy poca), como en un interruptor abierto. Sin embargo, si ambas de esas uniones se polarizan directamente ($V_{BE}, V_{BC} > 0$) se afirma que el transistor estará en la región de *saturación*. El máximo flujo de corriente posible ocurrirá en ese caso.⁶

No es difícil imaginar que la contraparte de un transistor *nnp* es uno *pnp*, en el cual el sustrato es silicio tipo *p*. Todas las características son complementarias a las que se describieron para el transistor *nnp*. Los símbolos de circuito para ambos tipos se proporcionan en la figura 6.

En el diseño de circuitos lógicos con BJT, los transistores desempeñan el papel de interruptores controlados por corriente; la corriente de control es la de la base. Con cero corriente de base, el interruptor está abierto; con una corriente de base mayor que algún valor de umbral, el transistor se lleva hacia la saturación y el interruptor se cierra.

⁶ Hay otras dos regiones en las cuales puede operar el transistor: una unión polarizada directamente y una polarizada inversamente. El caso en el que $V_{BE} > 0$ y $V_{BC} < 0$ se conoce como la región *activa directa*. En esta región el dispositivo opera como un *amplificador*, siendo la corriente del colector β veces la corriente de base, donde β puede ser tan alta como 100. En esta región el transistor no es útil para los circuitos lógicos, de modo que nunca se alcanzan las condiciones sobre V_{BE} y V_{BC} .

Bibliografía

Referencias históricamente importantes

- George Boole, *An Investigation of the Laws of Thought*, Nueva York; Dover, 1954.
- Richard W. Hamming, "Error-Detecting and Error-Correcting Codes," *Bell Syst Tech J* 29, abril de 1950, pp. 147-160.
- E. V. Huntington, "Sets of Independent Postulates for the Algebra of Logic," *Trans Am Math Soc* 5, 1938, pp. 288-309.
- Claude E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans AIEE* 57, 1938, pp. 713-723.
- Claude E. Shannon, "The Synthesis of Two-Terminal Switching Circuits," *Bell Syst Tech J* 28, 1949, pp. 59-98.

Referencias anteriores importantes

Libros

- S. H. Caldwell, *Switching Circuits and Logic Design*, Nueva York; Wiley, 1958.
- J. Hartmannis y R. E. Stearns, *Algebraic Structure Theory of Sequential Machines*, Englewood Cliffs, NJ, Prentice-Hall, 1966.
- Zvi Kohavi, *Switching and Finite Automata Theory*, Nueva York; McGraw-Hill, 1978.
- Edward J. McCluskey, *Introduction to the Theory of Switching Circuits*, Nueva York, McGraw-Hill, 1965.
- Carver Mead y Lynn Conway, *Introduction to VLSI Systems*, Reading, MA; Addison-Wesley, 1980.
- Stephen H. Unger, *Asynchronous Sequential Switching Circuits*, Nueva York; Wiley, 1969.

Artículos

- David A. Huffman, "The Synthesis of Sequential Switching Circuits," *J Franklin Inst* 257, marzo, 1954; 257-303, abril de 1954. pp. 257-303.
- David A. Huffman, "A Study of the Memory Requirements of Sequential Switching Circuits," MIT Research Lab for Electronics Tech Report 293, abril 1955.
- Maurice Karnaugh, "A Map Method for Synthesis of Combinational Logic Circuits," *Trans AIEE* 73(9), parte 1, noviembre de 1953, pp. 593-599.
- Edward J. McCluskey, "Minimization of Boolean Functions," *Bell Syst Tech J* 35(6), noviembre de 1956, pp. 1417-1444.
- G. H. Mealy, "A Method for Synthesizing Sequential Circuits," *Bell Syst Tech J* 34, septiembre de 1955, pp. 1045-1079.
- E. F. Moore, "Gedanken Experiments on Sequential Machines," *Automata Studies Annals of Math Studies* 34, Princeton, NJ, Princeton University Press, 1956, pp. 120-153.
- W. V. Quine, "The problem of Simplifying Truth Functions," *Am Math Monthly* 59(8), octubre de 1952, p p. 521-531.
- John von Neumann, "Probabilistic Logic and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies, Annals of Math Studies* 34, Princeton, NJ, Princeton University Press, 1956, pp. 43-49.

Referencias contemporáneas (desde 1980)

- S. Brown y Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*, Nueva York; McGraw-Hill, 2000.
- D. J. Comer, *Digital Logic and State Machine Design*, 3rd ed., Philadelphia; Saunders, 1995.
- D. D. Gajski, *Principles of Digital Design*, Englewood Cliffs, NJ; Prentice Hall, 1997.
- R. H. Katz, *Contemporary Logic Design*, Redwood City, CA; Benjamin-Cummings, 1994.
- M. M. Mano y C. R. Kime, *Logic and Computer Design Fundamentals*, 2nd ed., Englewood Cliffs, NJ; Prentice Hall, 2000.
- C. H. Roth, *Fundamentals of Logic Design*, 4th ed., St. Paul, MN, West, 1992.
- J. F. Wakerley, *Digital Design: Principles and Practices*, 3rd ed., Englewood Cliffs, NJ; Prentice Hall, 2000.

Índice

A

abanico de entrada, 61-63, 65
 restricción en CMOS, 61-63
abanico de salida, 64-65
ABEL
 declaraciones de sección, 292
 declaraciones, 277, 278
 descripción del comportamiento en, 292
 ecuaciones, 277, 278
 especificaciones jerárquicas, 288
 niveles de jerarquía, 288
 operadores, 278, 282
AC, 318
acarreo, 9
 de entrada, 126
 de desbordamiento, 128
 de salida, 9, 126
 generado, 129, 130
 propagado, 129, 130
Activación por flanco, 161, 171
activo en nivel alto, 56
activo en nivel bajo, 56
acumulación de productos parciales, 311
acumulador, *Véase* AC adyacencia
 geométrica, 79, 80
 lógica, 79, 83
afirmación de alto o bajo, 56-57
ajuste de un latch, 163
alfabeto, 4
álgebra booleana de dos elementos, 37
álgebra booleana, 32
 axiomas, 32
 consistencia de, 32
 dominio de, 32, 37
 elemento identidad
 respecto a +, 35
 respecto a Σ , 35
 elementos de, 32
 operaciones en
 binarias, 33
 complementación, 33
 multiplicación, 33
 suma, 33, 37
 unarias, 33
 postulados de Huntington, 33, 37
 principio de dualidad, 34
 teoremas de, 32
 absorción, 35

 consenso, 36
 idempotencia, 35
 involución, 35
 ley asociativa, 36
 ley de De Morgan, 36
 ley nula, 34-35
 simplificación, 35
álgebra de conmutadores, 37, 52-53, 55
álgebra
 booleana, 32
 de dos elementos, 37
 dominio de, 37
 de conmutaciones, 37
algoritmo Quine-McCluskey, 105-106
 función incompletamente especificada, 109
ALU, 318
análisis, 97
analizador lógico, 2
AND cableada, 71
aritmética binaria, 9
 división, 10
 multiplicación, 9
 resta, 9
 suma, 9
arquitectura Harvard, 322
arquitectura Princeton, 322
arquitecturas de microcontrolador, 335-336
arreglo de compuertas programables en campo.
 Véase FPGA
arreglo lógico programado. *Véase* PLA
asignación de estados, 198
asignación de tecnología de especificaciones
ABEL, 302, 304-306
ASM, 221-225
 caja
 de decisión, 221, 222
 de estado, 222
 de salida condicional, 222
 diagrama, 221-227
 para la máquina de Moore, 221-222

B

Base
 del sistema decimal hexadecimal, 5
 del sistema numérico binario, 5
 del sistema numérico decimal, 5
 del sistema numérico octal, 5
BCD, 17

- bit (dígito) más significativo, 5
- bit (dígito) menos significativo, 5
- bit de paridad, 21
- bits de mensaje, en el código Hamming, 23
- bits de verificación de paridad en el código Hamming, 23
- BJT, 58, 342
- bloque lógico configurable (CLB), 300
- bloques de estados, 210
- bms, 11
- borrado asíncrono (CLR), 178
- burbuja de inversión, 98
- burbuja, 53
- bus, 280

C

- CAD, 275
- cambio de entrada permisible, 245
- capacitancia, 61
- características de los circuitos de compuerta, 97-98
- carga
 - de compuertas, 57
 - de registro, 176
- carrera, 165-166, 168
- carrera crítica contra no crítica, 257-258, 260
- Celda con 0, (celda, cuadro o cuadrado) (que contiene, con)
 - cero, 85
- Celda con 1, (celda, cuadro o cuadrado, casilla o compartimiento)
 - (que contiene, con) uno, 85
- celda de memoria, 159
 - primitiva, 159, 162
- CI de aplicación específica, 69
- ciclo, 259, 260-261
- ciclo búsqueda-decodificación-ejecución, 317, 320-321
- ciclo de trabajo, 167
- cierre, 33
- circuito asíncrono en modo fundamental, 242, 246
- circuito de conmutación, 57
- circuito electrónico, 4
- circuito integrado (CI), 4, 58, 66-69
 - características, 67-68
 - confiabilidad, 67
 - de aplicación específica (ASIC), 69
 - etapas de entrada-salida
 - de compuertas lógicas CMOS, 60-61
 - de compuertas lógicas TTL, 61-62
 - requerimientos de potencia, 67
 - tamaño y espacio, 66
- circuito lógico electrónico, 339
- circuito secuencial, 159
 - asíncrono, 187, 241, 243
 - primitivo, 243
 - carrera en, 257-258
 - con reloj, 187
 - síncrono, 187
- circuito
 - de salida múltiple, 112
 - electrónico, 4
 - integrado, 4
 - lógico,
 - análisis, 196-197
 - combinacional, 125
 - costo del diseño, 68
- circuitos de salida múltiple, 112
- circuitos secuenciales retroalimentados, 241
- circuitos secuenciales síncronos, 187
- CMOS, 58, 291, 294
 - familia de compuertas, 59
 - compuerta, 60
- cobertura cerrada mínima, 251-253
- cobertura de un grupo rectangular de orden k , 88
- cobertura, mínima cerrada, 251-252, 253
- cociente, 7
- codificación, 20
- codificador, 139
- código,
 - alfanumérico, 19-20
 - cíclico, 18
 - de corrección de errores, 22, 23
 - de detección de errores, 21
 - de siete segmentos, 19
 - Gray, 18
 - Hamming, 23, 193
 - bits de mensaje en, 23
 - síndrome en, 23
- códigos, 15
 - alfabético, 15
 - alfanumérico, 19
 - ASCII, 19-20
 - cíclicos, 18
 - de corrección de errores, 21, 22-23
 - de detección de errores, 21
 - bit de paridad, 21
 - decimal codificado en binario (BCD), 16
 - Gray, 18
 - Hamming, 23-25
 - de autocomplementación, 17, 19
 - de siete segmentos, 19
 - ponderados, 16-18
- combinación XOR-AND, 127
- Comparador del bit 2, 4, 102, 104
- Comparador
 - de números de 1 bit, 51, 55
 - de números de 2 bits, 102-103
 - de números de 4 bits, 104
 - descripciones ABEL de, 281-282

- de números pares de bits, 105
- de números impares de bits, 105
- compatibilidad, 277
- complemento a diez, 11, 12
- complemento de dos, 11, 13
 - sumador/restador, 132-133, 133
 - representación numérica, 12-13
- complemento, 33, 38
 - a diez, 11, 12
 - a dos, 11, 12
 - a uno, 11
 - sumador/restador, 133-134
- componente, 3
- compuerta AND, 53
 - salida de, 139
- compuerta de colector abierto, 71
- compuerta de drenaje abierto, 71
- compuerta de gobierno, 167
 - de latch maestro-esclavo, 169
- compuerta lógica de tres estados, 69
- compuerta lógica, 52-53, 54-55
 - aspectos prácticos relativos a, 57
 - características de entrada salida de, 59
 - colector abierto, 70
 - drenaje abierto, 70
 - ideal, 54
- compuerta NAND, 53
 - forma alternativa de, 53-54
 - operación, 50-51
- compuerta NOR, 53
 - forma alternativa de, 53-54
 - operación, 50-51
- compuerta NOT, 57
 - operación, 39
- compuerta OR, 52
- compuerta XOR, 50-51, 52-53, 55
 - como comparador de números de 1 bit, 55
 - estructuras alternativas, 54
- compuerta. Véase compuerta lógica
- computadora
 - conjunto complejo de instrucciones (CICS), 332-333
 - conjunto reducido de instrucciones (RISC), 334
 - digital, 1-3
 - memoria, 317
 - programa almacenado, 317
- comunicación asíncrona, 226-227
- conjunto de operaciones,
- conjuntos compatibles, 248
 - máximos, 254
- constante de conmutación, 37
- consumo de potencia, 59
- contador, 187
 - ascendente-descendente, 219-221
 - de anillo trenzado, 217-219

- de anillo, 217-219
 - estado de corte en, 218-219 *Hang up*
 - diseño con autocorrección, 219
- de distancia unitaria, 216-219
- de modo múltiple, 219-221
- de modo sencillo, 215-216, 219-220
- de módulo, 3, 6, 181, 216-218
- de programa, 318
- multimodo, 219-220, 221
- número de módulo de, 215
- contadores síncronos, 215-221
 - código asignado a estados, 217
 - distancia unitaria, 216
 - modo sencillo, 215
 - número de módulo de, 215
- control de carga de registro de corrimiento, 177-178
- conversión
 - base de, 6
 - de base, 7
 - de código, 142-143
 - de hexadecimal en binario, 8-9
 - de números
 - binario a decimal, 6-7
 - decimal a binario, 6-7
 - hexadecimal a binario, 8
 - octal a binario, 8
 - octal a decimal, 7
 - hexadecimal a binario, 8
 - octal a binario, 8
- convertidor de código, 142
- convertidor serie-paralelo, 212
- corriente de sumidero, 63
- Corrimiento de barril, 290-291
- costo de empaque, 68
- Costo del diseño, 68
- CPLD, 295, 297
- CPU, 318, 319-320
- cristales líquidos, 19

D

- decimal codificado en binario. Véase BCD
- transistor de unión bipolar. Véase BJT
- decodificador, 139
 - como circuito lógico de propósito general, 142
 - de árbol, 141-142
 - de estado, 188, 198-199
 - de exceso 3 a siete segmentos, 143
 - de n a 2^n líneas, 139-141
 - dígito (bit) más significativo de, 5
 - dígito (bit) menos significativo de, 5
 - de salida, 188, 189, 198
- demostración
 - por contradicción, 34
 - por inducción matemática, 34

demultiplexor, 134, 139-140
 circuito, 140
 diagrama de estado, 207
 dependencia de datos, 331
 desbordamiento, 132
 aritmético, 132
 descripción a nivel de transferencia de registro.
Véase RTL
 Descripción ABEL
 de comportamiento, 279, 292
 de detector de secuencia, 282-283
 del corrimiento de barril, 290, 291
 estructural, 287
 operativa, 279
 descripción de comportamiento, nivel de
 transferencia de registros (RTL), 308
 descripción estructural, 308
 en ABEL, 287
 designación de señales en ABEL
 nodo, 282
 patilla, 282
 detector de cambio de nivel, 202, 203
 detector de errores, 204-205
 diagrama de bloques del *latch* con reloj, 167
 diagrama de estado, 189-190, 207
 de la unidad de control, 314
 diagrama de temporización, 99
 Diagrama del 74LS90, requerimientos de
 excitación, 315
 diagrama esquemático, 57
 dieléctrico de un condensador, 61
 dígito que se lleva, 10, 12
 directivas, 303
 diseño asistido por computadora. *Véase CAD*
 dispositivo biestable, 163
 dispositivo lógico programable complejo. *Véase CPLD*
 Dispositivos de E/S, 325
 dispositivos lógicos programables. *Véase PLD*
 dispositivos primitivos, 66
 disputa de bus, 324
 distancia entre palabras de código, 21
 en ABEL, 283-284, 285
 división de estados, 210

E

enlace en un PLA, número de, 148
 Entrada
 de ajuste de latch SR,
 de puesta a cero (CLR), 178
 de puesta a cero del latch SR, 163
 de puesta a uno (PR), 170, 178
 primaria, 159
 secundaria, 159

entradas asíncronas, 226
 entradas de nivel, 242
 entrada-salida. *Véase E/S*
 EPROM, 300
 equivalencia de estado, 207
 estabilidad
 de forma de onda de reloj, 161
 de frecuencia de reloj, 161
 estado(s)
 compatibles, 247, 251
 de puesta a cero, 109-191
 de sucesor con X, 208
 del sucesor con 1, 208
 distinguibles, 208
 estructura, 309
 distinguibles de longitud k , 208
 cobertura, 88
 equivalente de longitud k , 209
 equivalentes, 208, 209
 estable, 242
 indeterminado en contador de anillo,
 218-219
 indistinguibles, 193
 inestable, 242
 interno, 243
 par implicado de, 248
 presente, 187, 188
 siguiente, 188
 estructura del sistema digital, 309
 estructura regular, 309
 exactitud funcional, 275
 expresión, 42
 de conmutación, 40
end, 209 (se refiere a una palabra reservada)
 irreducible, 42, 87
 mínima, 87, 91
 module, 290
 producto de sumas, 49
 redundante, 41
 suma de productos, 77
 mínima, 89
 Expresiones ABEL
 comentario, 277
 declaración, 278
 istype, 278
 módulo, 278

F

familia XC4000, 299, 300
 familias lógicas, 58
 flip-flop, 55, 162-163, 171, 187
 asíncrono, 174
 biestable (T), 174
 biestable (T). *Véase flip-flop T*

D activado por flanco, 171-174,
de retardo (*D*). Véase flip-flop D
indicador dinámico, 176
JK, 174
requerimientos de excitación de, 175
retardo (*D*), 171-172
T, 174
flujo de diseño, especificaciones HDL de, 301, 302
forma canónica, 41, 42, 47
 de máquina de memoria de entrada finita, 212
 de máquina de memoria de salida finita, 213
 producto de sumas, 42, 47, 78
 suma de productos, 42, 47, 78
forma de producto de sumas, 42, 49, 50, 78
 irreducible, 91
 expresión mínima, 91
generador de paridad, 193
forma de suma de productos, 42, 47, 49, 77
 irreducible, 87
 no redundante, 42
 forma de suma de productos, 77, 86
formato electrónico de intercambio de datos
 (EDIF), 302
FPGA, 299, 300
frecuencia de reloj, 160-161
 ciclo de trabajo de, 171
Función
 de conmutación, 45, 47, 48
 realización mínima de, 87
 especificada incompletamente, 100-101
fusible en una ROM, 145

G

gráfico lineal, 189
grupo rectangular de orden k , 85, 89, 91, 106

H

habilitación, 135
hardware, 57
 desventaja, 131
HDL, 301-302
Herramienta de síntesis ABEL, 281
herramientas de diseño asistido por computadora,
301
herramientas de síntesis, 276

I

idempotencia, 35
implementación, 37
 AND-OR, 92
 de expresiones lógicas, 94-95
 dos niveles, 92-93

NAND, 93
OR-AND, 94
implementar, 555, 92
implicante, 88-89
 primo, 88-89, 126
 determinación de, 105-109
 esencial, 89, 90
incompatibilidad, 249
indicador dinámico, 171, 176
inducción perfecta, 39
ingreso del diseño mediante diagrama
esquemático, 275
integración a pequeña escala. Véase SSI
interruptor, 6
inversor, 53, 162-163
involución, 35

J

jerarquía de memoria, 332

L

latch
 activado por pulso, 170
 con compuerta, 172
 esclavo, 169
 JK, 168
 maestro-esclavo, 168, 169, 170
 activado por pulsos, 170
 conjunto compatible máximo de estados,
 251, 283
 entrada de puesta a uno, 170
 entrada de puesto a cero, 170
 SR, 163-167
 transparencia, 167
latch *SR*, 162-167
 con reloj, 166-167
 tabla de transición, 167
 diseño de compuerta NOR, 166-167, 169
LED, 19
Lenguaje Avanzado de Expresión Booleana.
 Véase ABEL
lenguaje de descripción del hardware. Véase HDL
lenguaje de máquina, 319
ley
 asociativa, 36, 53
 conmutativa, 33, 34
 de absorción, 35, 85
 de De Morgan, 36-37, 51, 53
 forma dual de, 43
 generalización, 43, 44
 distributiva, 33, 84
 nula, 34-35
leyes booleanas, 41

líneas de palabras en ROM, 144
 literal, 41, 42, 49, 87
 lógica,
 aleatoria, 310
 cableada, 69
 de arreglo programado. *Véase* PAL
 de proposiciones, 37, 49, 56
 del arreglo programable. *Véase* PAL
 mezclada, 55-56
 negativa, 55-57
 positiva, 55-57
 transistor-transistor, 58
 longitud de palabra, 11
 LSI, 66

M

macrocelda, 291
 mapa de asignación, 205
 mapa de Karnaugh, 81, 83
 mapa. *Véase* mapa lógico
 mapa lógico, 79-84
 cíclico, 90
 máquina con memoria finita de entradas, 211-212
 máquina con memoria finita de salidas, 213-214
 máquina con rango de memoria finita, 211
 máquina de estado algorítmica. *Véase* ASM
 máquina de estados finitos, 211, 213-214, 311
 máquina de Mealy, 160, 188, 192, 194, 202
 procedimiento de diseño, 201
 máquina de Moore, 160, 188, 191, 201
 máquina especificada incompletamente, 247-248
 maxitérmino, 41, 43, 46, 77, 82
 distinguible, 91
 lista, 76, 78
 memoria, 3, 159
 caché, 332
 escritura en, 160
 finita de entradas, 211-212
 finita de salidas, 213
 lectura de, 160
 registro de dirección. *Véase* MAR
 sólo de lectura. *Véase* ROM
 metal-óxido-semiconductor, 340
 métrica retardo-potencia, 59
 Microcontrolador 8051, 335
 microinstrucciones, 328-329
 microoperaciones, 320
 Microprocesador
 arquitecturas, 329
 800 (Motorola), 325
 8085 (Intel), 325
 Pentium, 333
 minimización de máquina, 207, 209-210
 minitérmino, 41, 43, 46-47, 77, 82

distinguible, 90, 91
 índice de, 107
 lista, 76-77
 modelo
 de Mealy, 160
 de Moore, 160
 MOSFET, 60, 291, 340-341
 como un interruptor controlado por voltaje, 60
 como una fuente de corriente, 60
 modo de acrecentamiento, 340
 multiplexor (MUX), 134-139, 286-287
 como circuito lógico de propósito general,
 136-137
 dual de cuatro entradas, 136
 multiplicación, 33
 lógica, 37
 multiplicador de suma y corrimiento, 310, 311-
 313, 316

N

negación, 38
 nodo en ABEL, 276
 nodo en un gráfico, 189-190
 NOR exclusiva, 50-51
 compuerta, 52
 número binario, 4, 13
 representación de entero decimal, 8
 sistema, 4
 número de módulo de contador, 215, 220
 números
 binarios, 4, 5-6
 representación en complemento a dos de,
 11, 12
 representación en complemento a uno de,
 11, 12
 parte fraccionaria de, 4
 parte integral de, 4
 rango de, 13
 complemento a dos, 12
 de posición 4
 decimal, 4, 5-6
 hexadecimal, 5-6
 octal, 5-6
 representación mezclada de, 12

O

operación
 AND, 38
 binaria, 33
 OR, 39
 unaria, 33
 universal, 51
 compuertas, 97-98

operaciones completas funcionalmente, 51
 operaciones de conmutación
 AND, 38
 NOT, 39
 OR, 39
 OR exclusiva, 50
 compuerta, 52
 operación, 133
 oscilación, 259
 oscilador, 161
 oscilador de cristal de cuarzo, 161

P

PAL, 148-150, 290-291
 PAL16L8, 290-294
 palabras de código, 21
 PALCE16V8, 291, 293-295, 296, 297, 301
 macrocelda de salida de, 295, 296
 paridad de palabras de código, 21
 partición de equivalencia, 210
 pastilla de circuito integrado, 58, 66
pin (en ABEL), 276 (palabra reservada)
 pipeline, 329-331
 PLA, 146-148, 290-291, 327, 328
 programable en campo, 148
 programable por máscara, 148
 tabla de programación, 147
 PLD XC9500, 295, 296, 297, 302
 PLD, 68, 146, 291
 postulados de Huntington, 33
 principio de dualidad, 33-34, 43
 programación de una ROM, 145
 protocolo, 226
 de intercambio, 226
 prototipo, 275
 prueba de existencia del teorema de Shannon, 48-49
 puesta a cero de un latch, 165-166
 puesta a uno asíncrona (PR), 178

R

rango de memoria, 211, 213
 finita, 213
 realización, 55
 reducción de máquinas incompletamente
 especificadas, 247-248
 registro, 176, 206-207
 carga de, 176
 de corrimiento, 178-179, 211-212, 311
 carga en paralelo, 177-178
 control de carga, 177-178
 conversión paralelo-serie, 178
 entrada paralelo, salida paralelo, 170
 salida en paralelo, 177

 salida en serie, 176
 universal, 180-181
 de dirección de memoria (MAR), 319, 321, 322
 lectura de, 176
 universal, 180-181
 reglas de asignación de estado, 198-199
 reglas de asignación, 199
 relación de equivalencia, 51
 relevador, 6
 requerimientos de excitación de flip-flop, 175-176
 residuo, 7
 restador binario, 132-133
 retardo de una compuerta
 como función de la capacitancia de carga, 65
 extrínseco, 65
 intrínseco, 65
 retraso de propagación, 126, 128, 131
 riesgo, 99, 100
 de temporización, 99
 dinámico, 265
 esencial, 265-267
 estático, 261-265
 ROM, 145-146, 327, 328
 borrable, 144-145
 en blanco, 145
 programable en campo. Véase PROM
 RTL, 311

S

salida, 188
 secuencia aceptable, 189-190
 secuencia de entrada aplicable, 248
 secuencia de entrada, 190
 aplicable a un estado, 247
 secuencia distinguible, 208
 semiconductor, 339-341
 semisumador, 130
 señal
 continua, 3
 de extensión punto en ABEL, 283
 discreta, 3
 espuria, 99
 Shannon, Claude, 37
 símbolo esquemático, 53
 simplificación, 35
 simulación, 275, 301, 303-304
 de especificaciones ABEL, 304-306
 sin importancia, 100-101
 síndrome en el código Hamming, 23
 sistema
 analógico, 1, 2
 digital, 1
 numérico, 4, 5
 alfabeto, 5

base, 5
 conversión de base, 6-7, 8
 decimal, 5
 hexadecimal, 5
 octal, 5
 operativo, 3
 integrado (BIOS), 317
 sistemas algebraicos isomórficos, 37
 software, 3
 software Xilinx Webpack, 300
 SSI, 58, 66
 suma, 9, 33, 126
 binaria, 9
 de números binarios, 13
 lógica, 37
 sumador
 binario, 125-126
 completo, 126, 127, 129
 de acarreo anticipado, 128-129, 130, 131, 302
 de acarreo repetido, 128, 302
 especificación ABEL de, 277
 medio, 127
 sustracción, 12

T

Tabla,
 amalgamada, 248-251, 254
 de estados, 191, 192, 194, 202, 207
 de flujo primitiva, 243, 245
 de flujo, 243, 244-245
 primitiva, 243, 245, 254, 255
 reducida, 252-253
 de salida, 192
 de transición
 de estados, 195
 de latch SR, 165
 de máquina incompletamente especificada,
 253, 255
 de verdad, 38, 45, 46
 AND, 39
 OR, 39
 teorema de expansión de Shannon
 forma de producto de sumas, 49
 forma de suma de productos, 48
 prueba de la existencia del, 48
 tiempo de configuración, 171, 173, 226
 tiempo de retención de pulso, 171, 173, 226
 tope, 195
 transistor, 4, 339-341
 de efecto de campo (FET), 340
 de unión bipolar, 341
 trayectoria de datos, 308-309
 registros en, 309
 trayectoria lógica, 79-84

tren de pulsos periódicos, 161
 TTL, 58
 compuerta, 70, 71
 familia de compuertas, 58-59
 tubo de vacío, 57

U

unidad aritmética y lógica. *Véase* ALU
 unidad de control, 308, 309, 313
 descripción ABEL de, 317
 implementación
 cableada, 323
 especificación ABEL de, 323
 microprogramada, 322, 326-327, 328
 como máquina de estado, 310
 diagrama de estado de, 314
 unidad de procesamiento central, *Véase* CPU
 unidad funcional, 309
 unidades de hardware en paralelo, 331

V

valor verdadero, 37
 velocidad, 125
 verificación, 97, 196, 208
 verificador de paridad, 206
 verificador de paridad impar, 206
 Verilog, 276, 306
 vértice, 189
 VHDL, 276, 306
 VLSI, 66
 interruptor controlado por voltaje, 60
 nivel de voltaje, 55-56
 Von Neumann, John, 307

X

XC95144, 298
 Xilinx FPGA, 300
 XNOR, 50-51